

Neural Networks

Neural Networks



Artificial
Intelligence will...

Neural Networks



Artificial
Intelligence will...



Will the robots get
me?

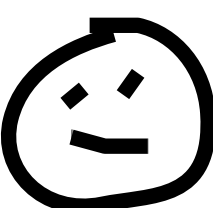
Neural Networks



Artificial
Intelligence will...



Will the robots get
me?



When can I
have a
bionic arm?

Neural Networks

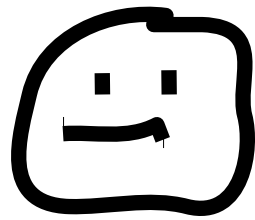
Artificial
Intelligence will...

Will the robots get
me?

When can I
have a
bionic arm?

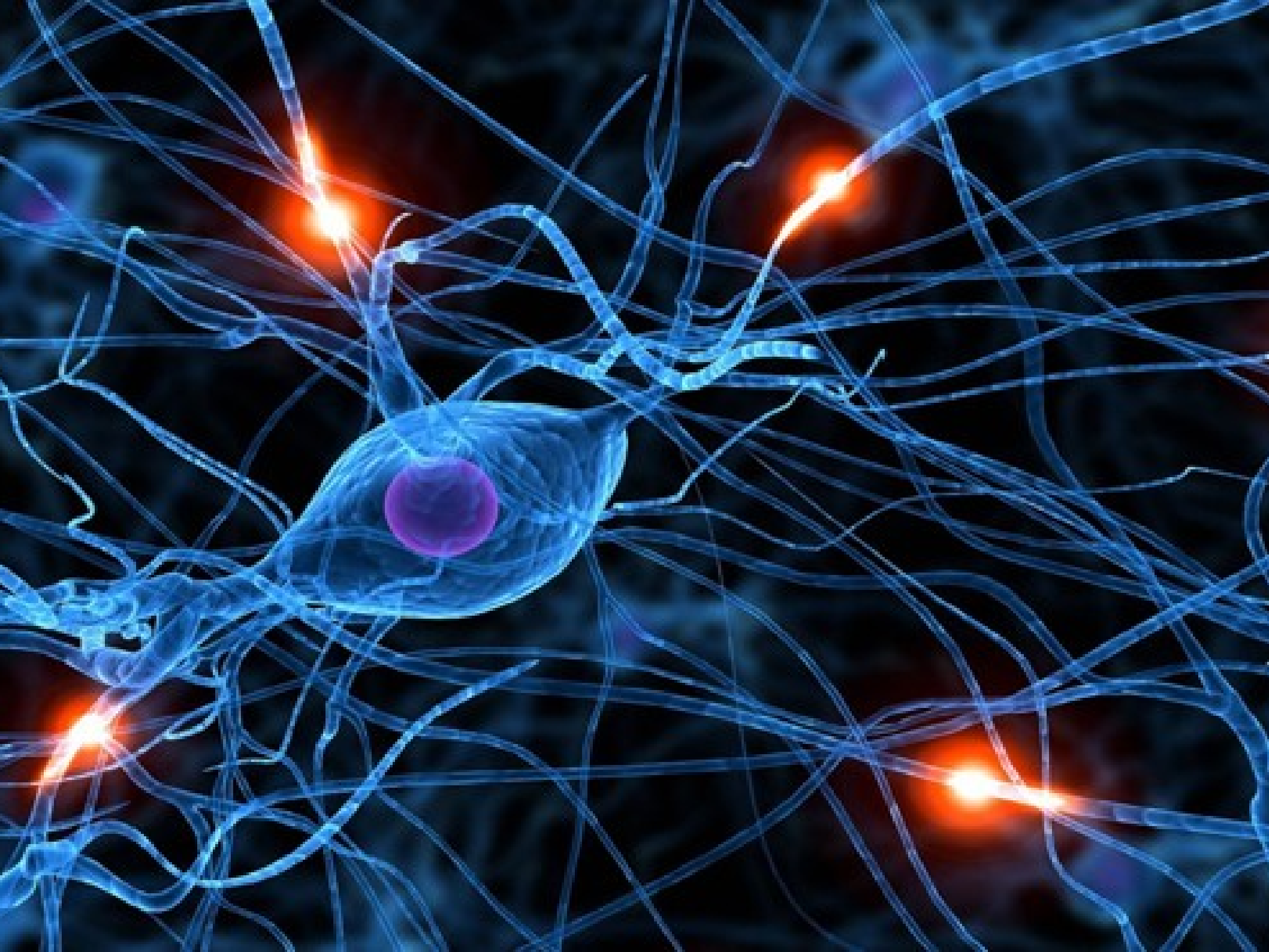
?

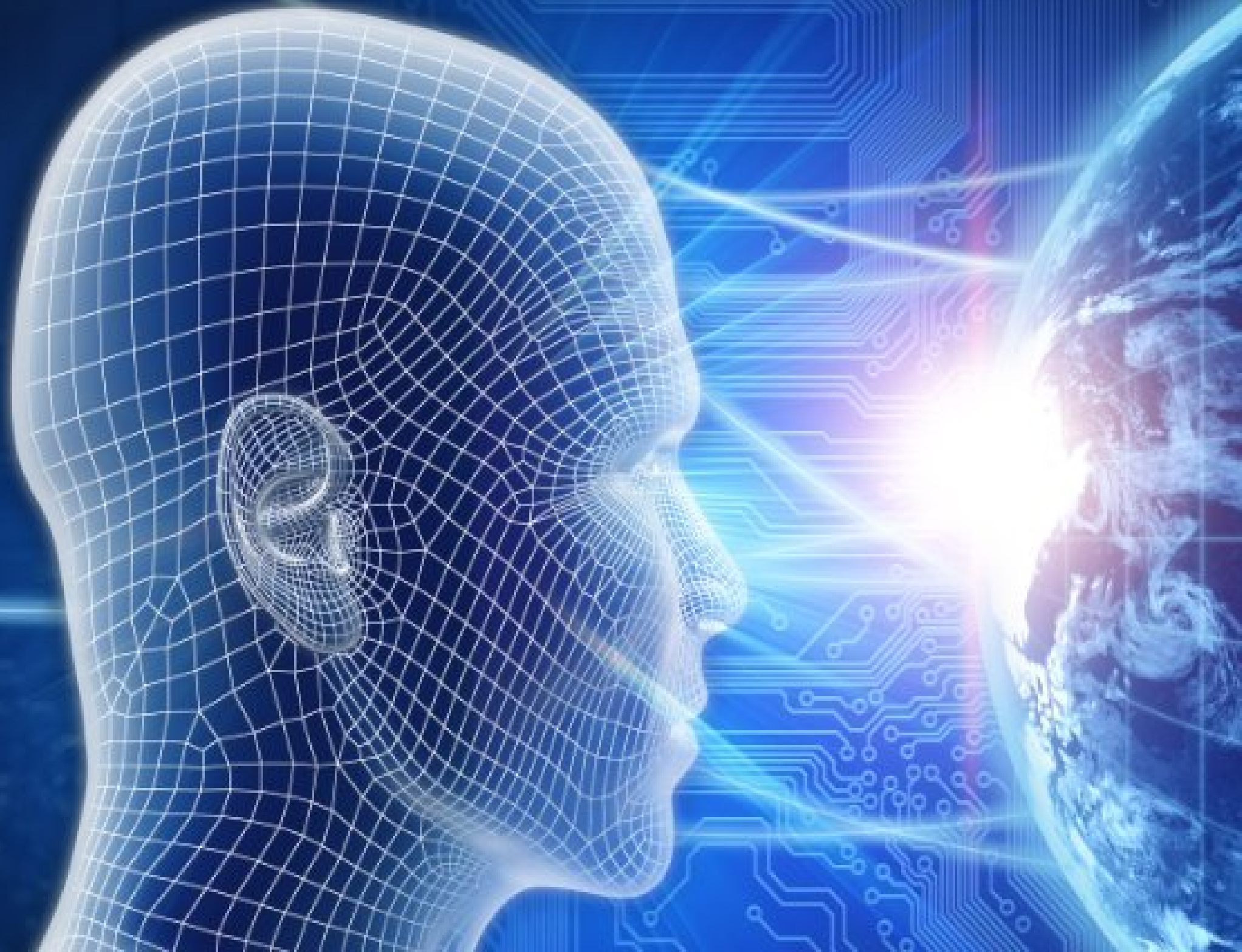
While bumping around the Internet reading about how AI and neural networks are bringing about our looming dystopian future or our blooming utopian society,

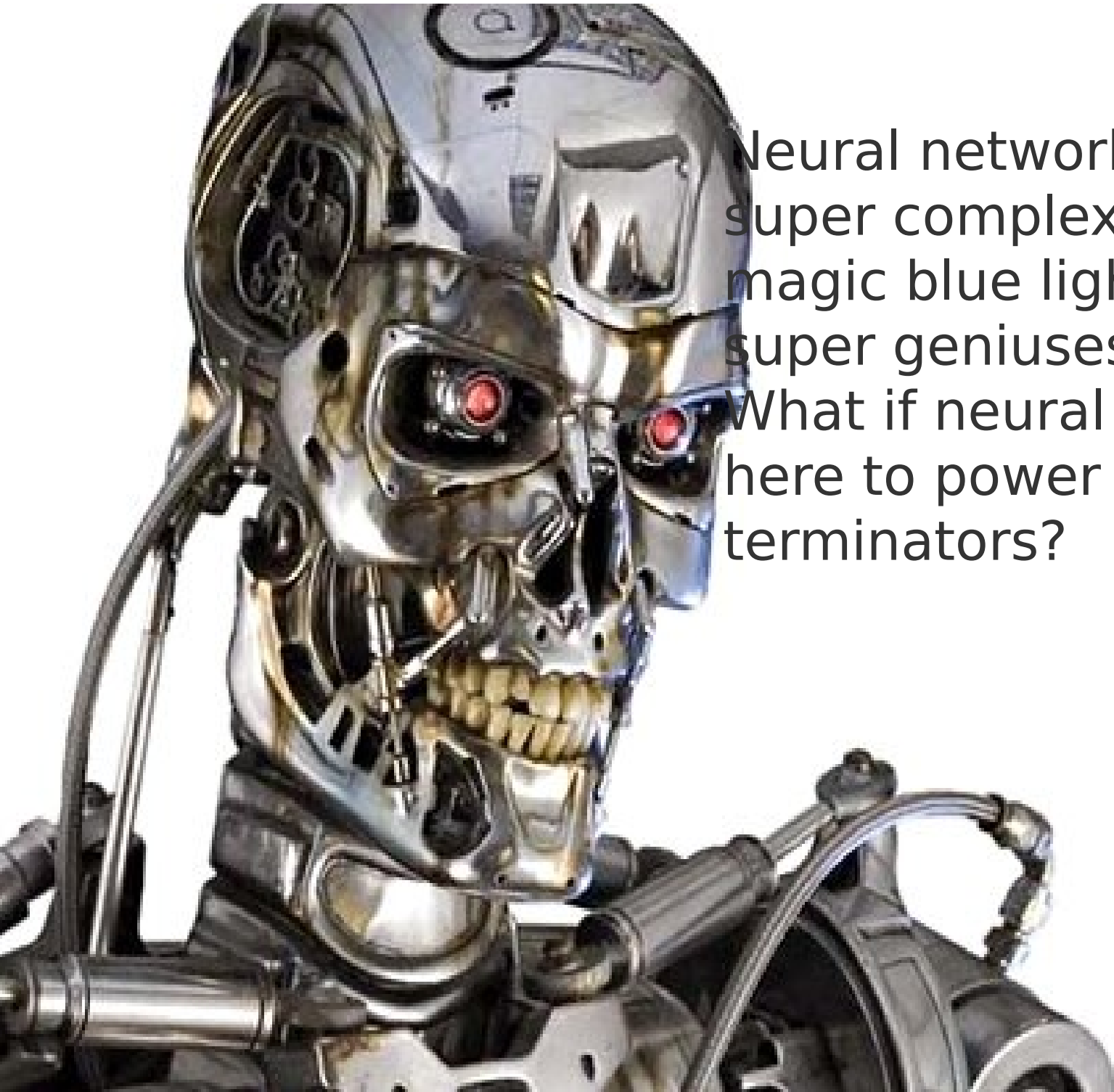


have you seen something like.....

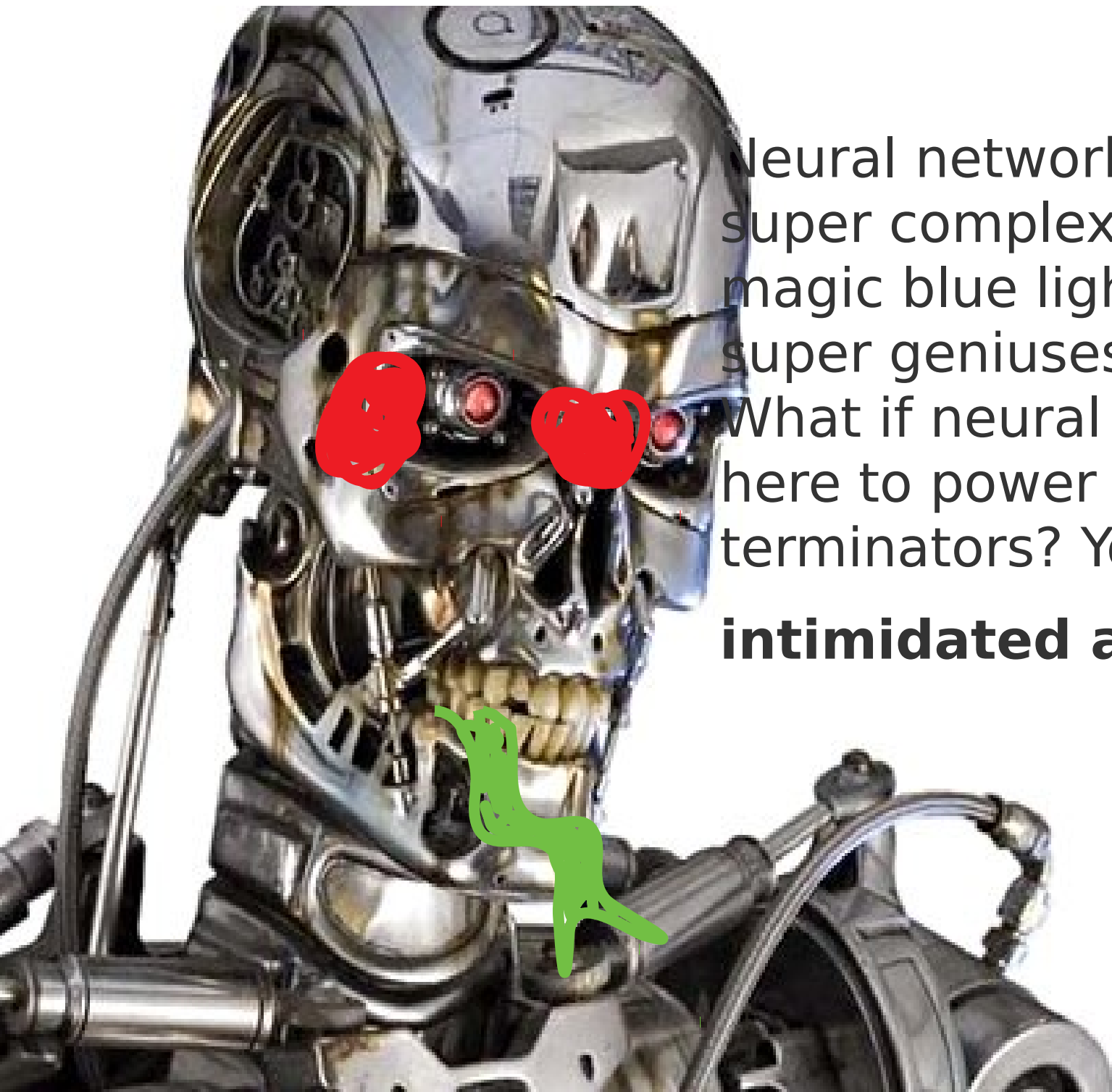








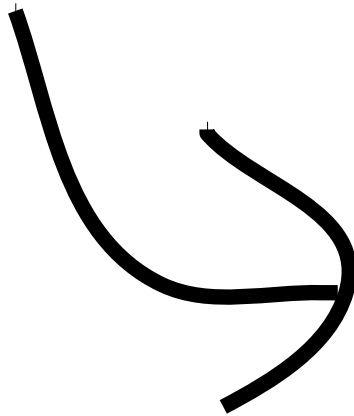
Neural networks seem like super complex bundles of magic blue lightning only super geniuses could make! What if neural networks are here to power future sentient terminators?



Neural networks seem like
super complex bundles of
magic blue lightning only
super geniuses could make!
What if neural networks are
here to power future sentient
terminators? You should be
intimidated and afraid!

Well, lets get something straight.

This is not a neural
network



Its Horse Crap

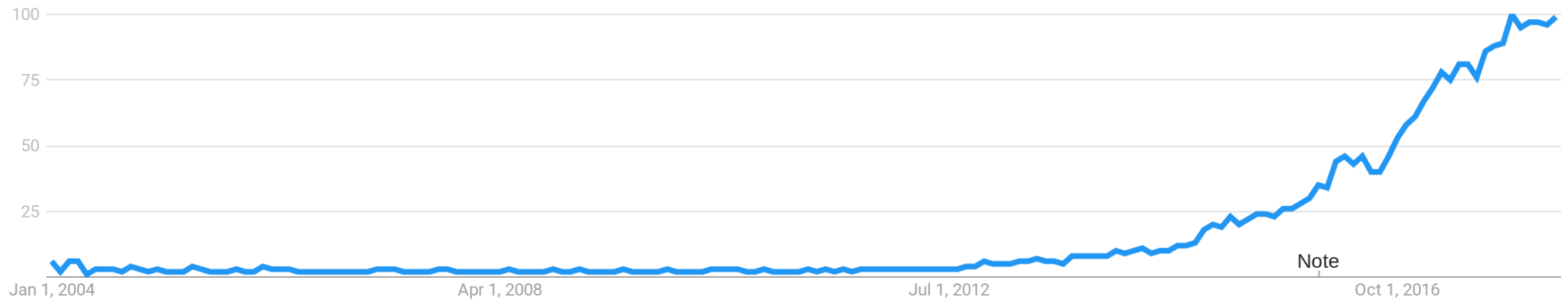


=



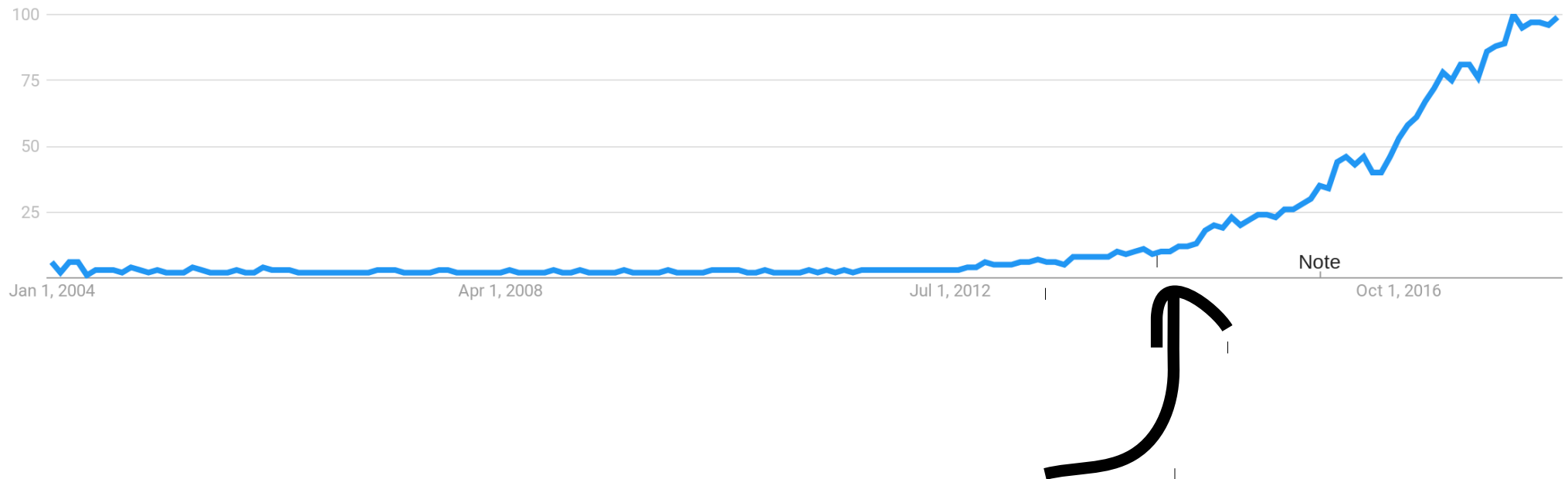
So, why is crap like this all over the Internet?

Google Search Trends: Interest in Deep Learning



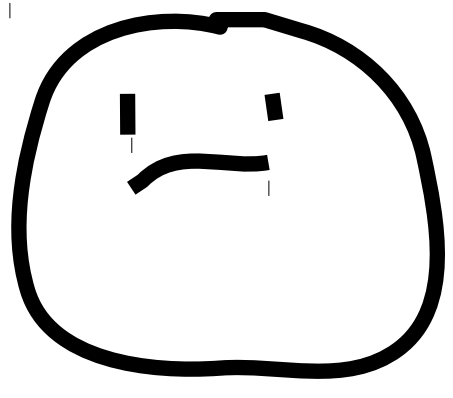
So, why is crap like this all over the Internet?

Google Search Trends: Interest in Deep Learning

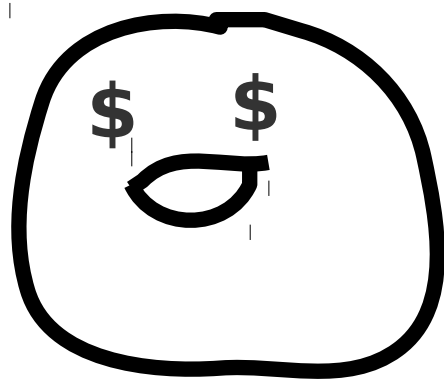


Recent advancements in the amount of accessible data and computation power have driven great progress and increased interest in different types of neural networks.

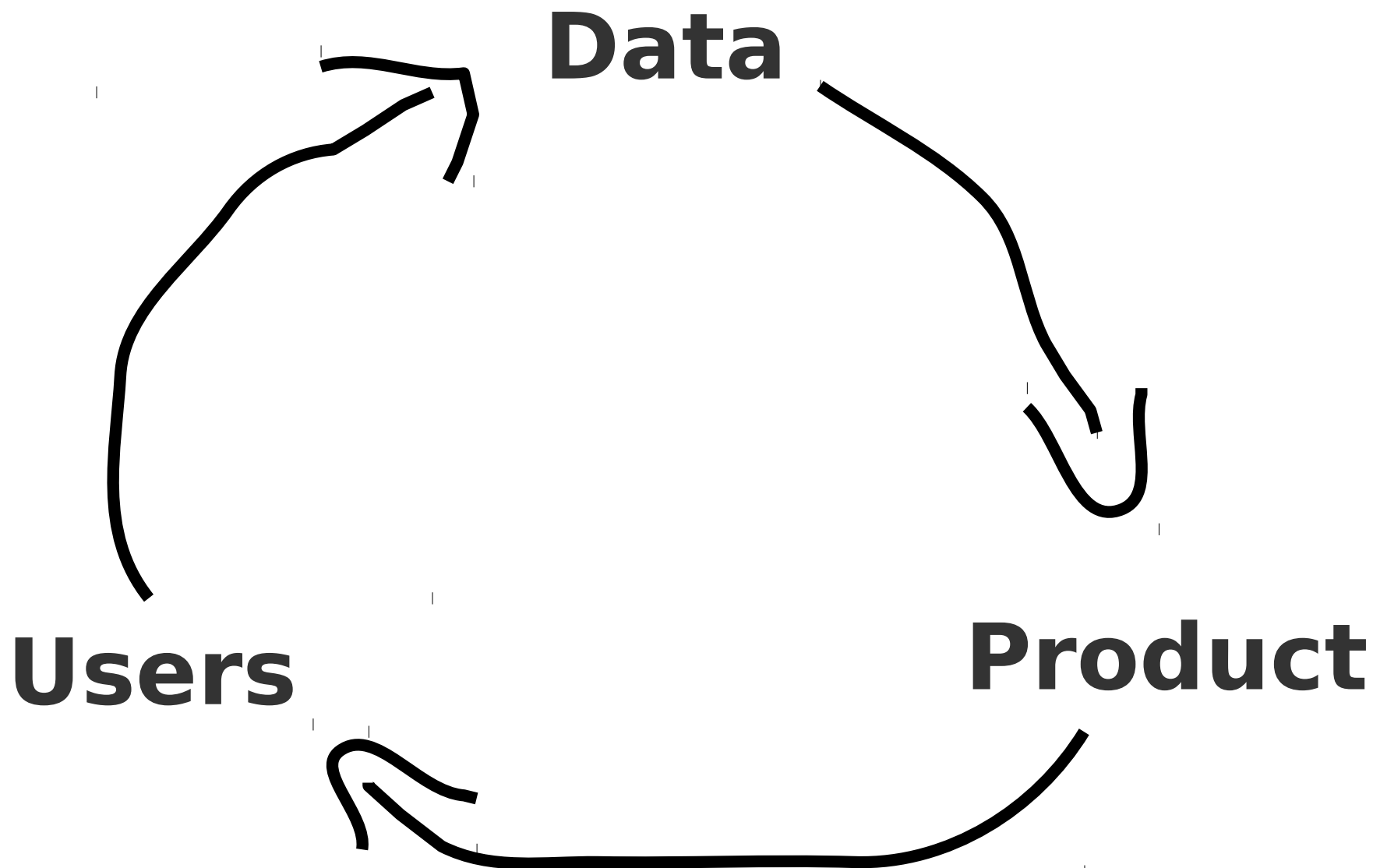
But why so much fuss?

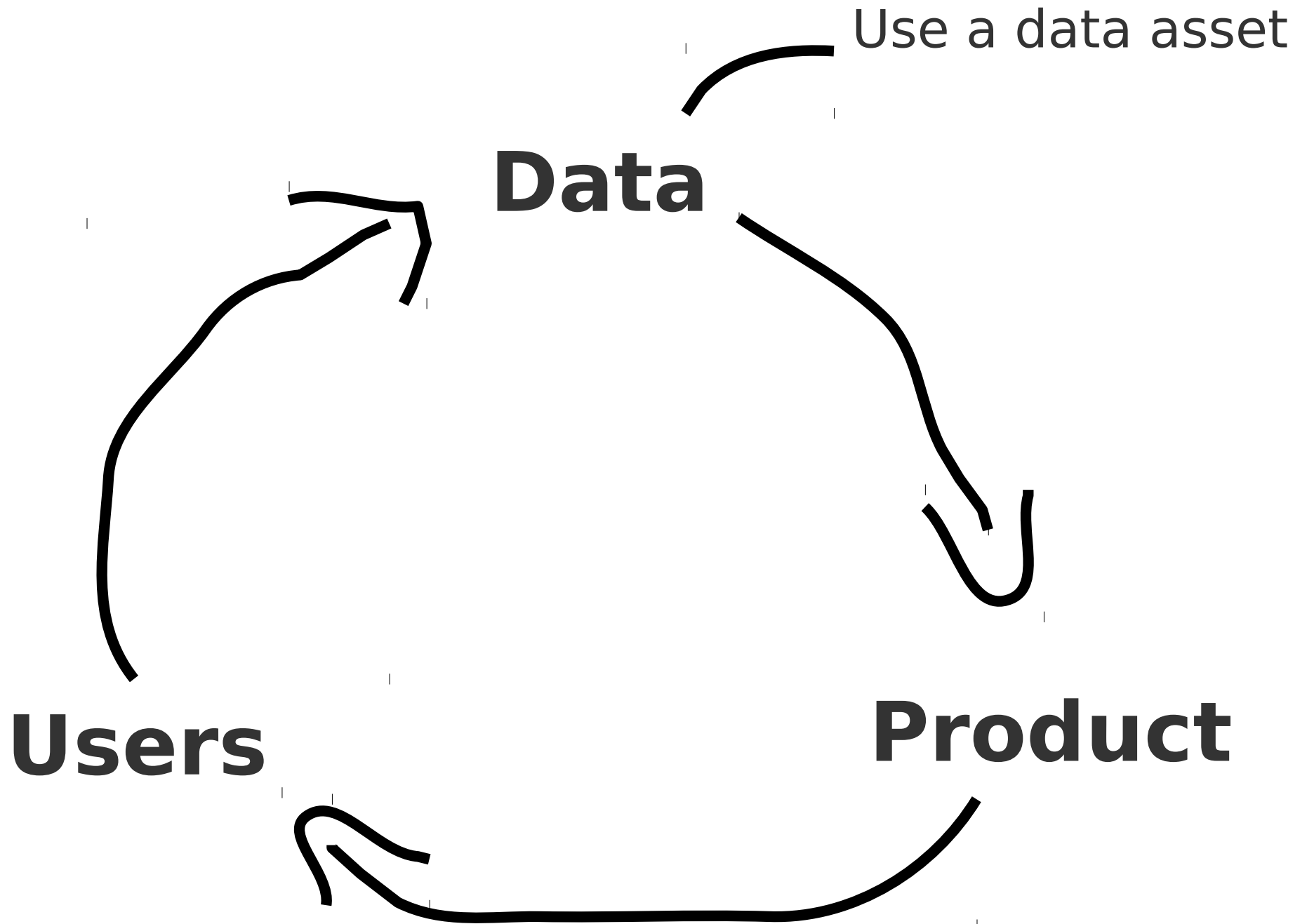


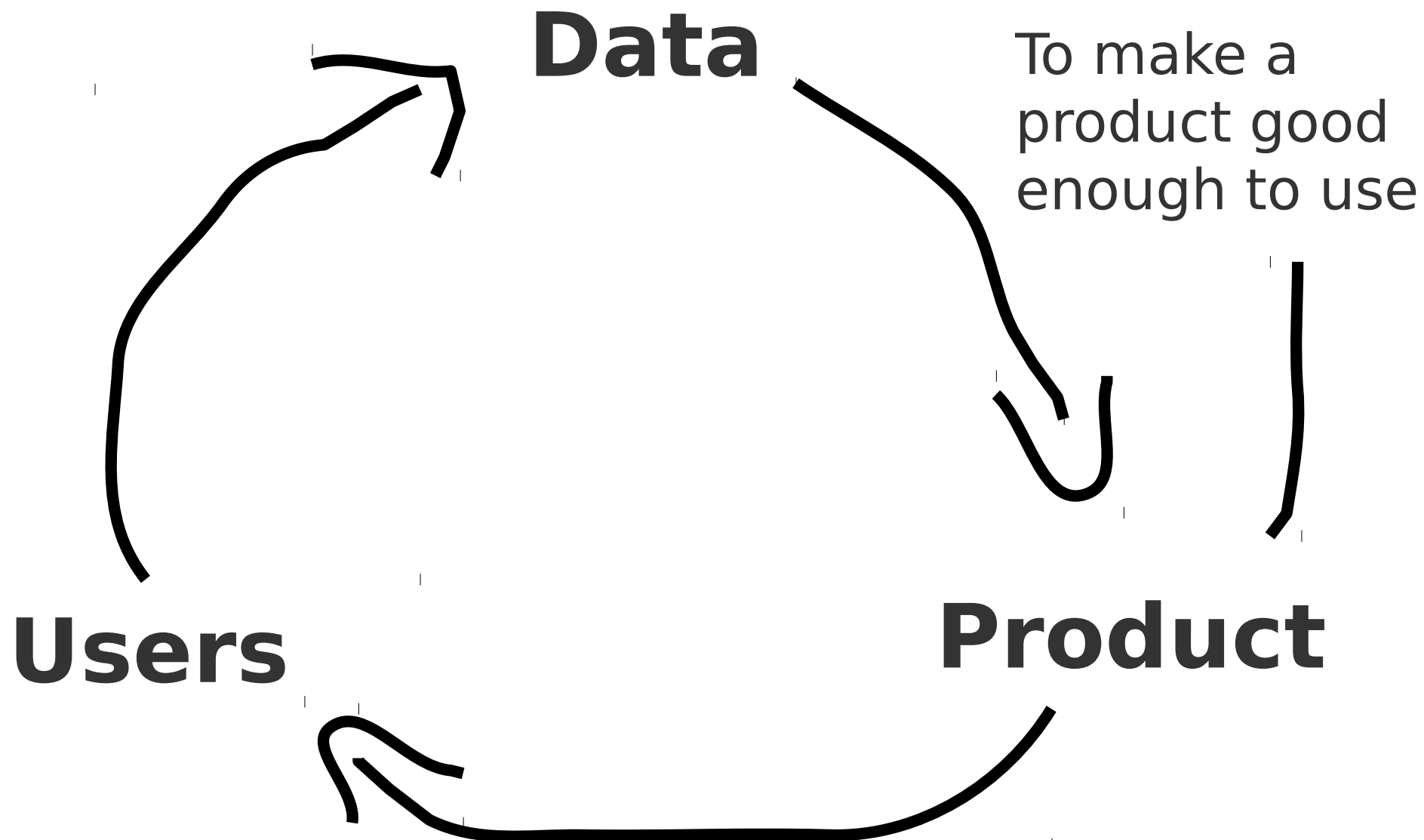
In the coming years, billions of dollars in revenue will be driven by AI technologies, according to fancy people.



Billions? How?





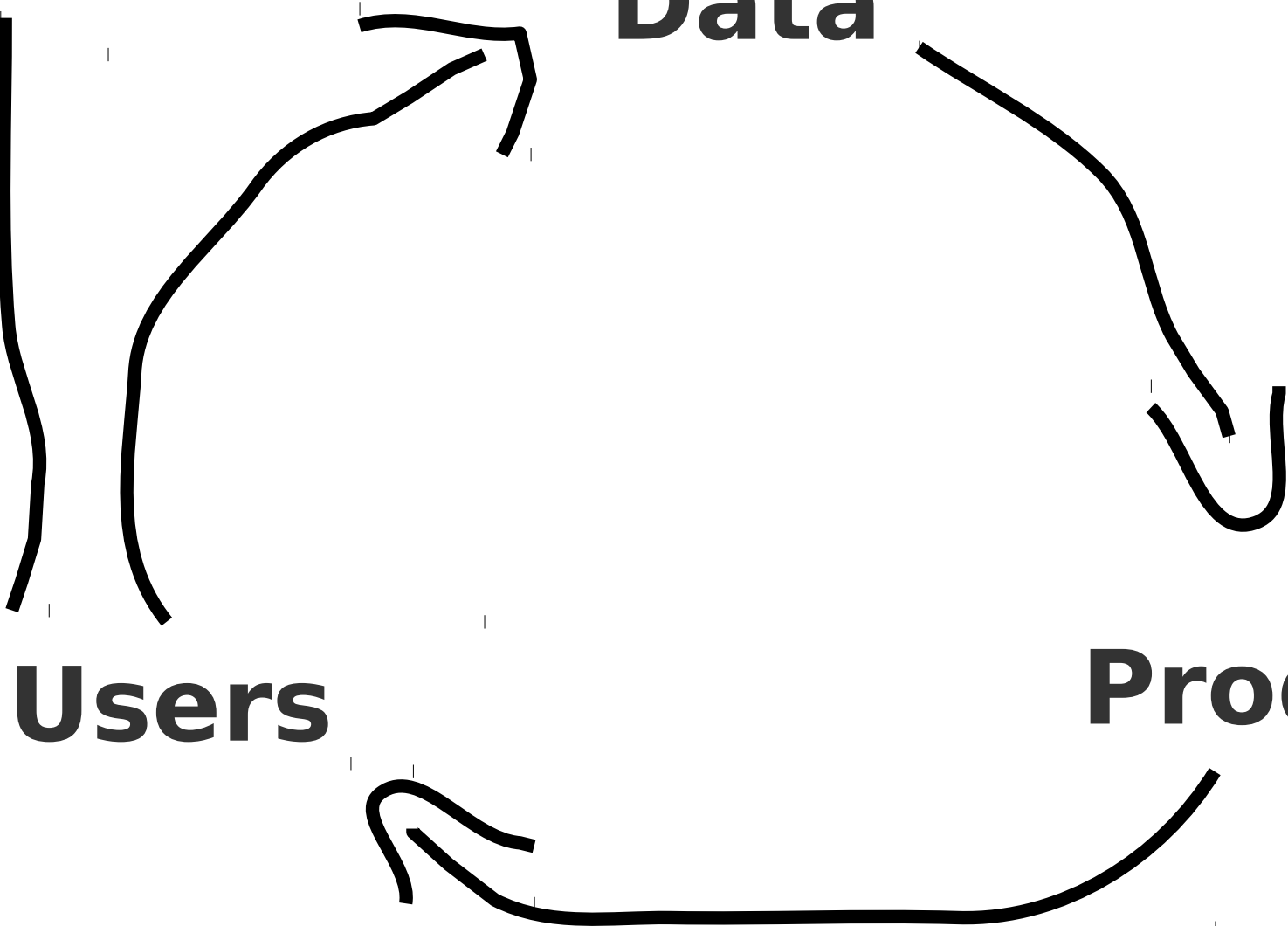


People use it

Data

Product

Users

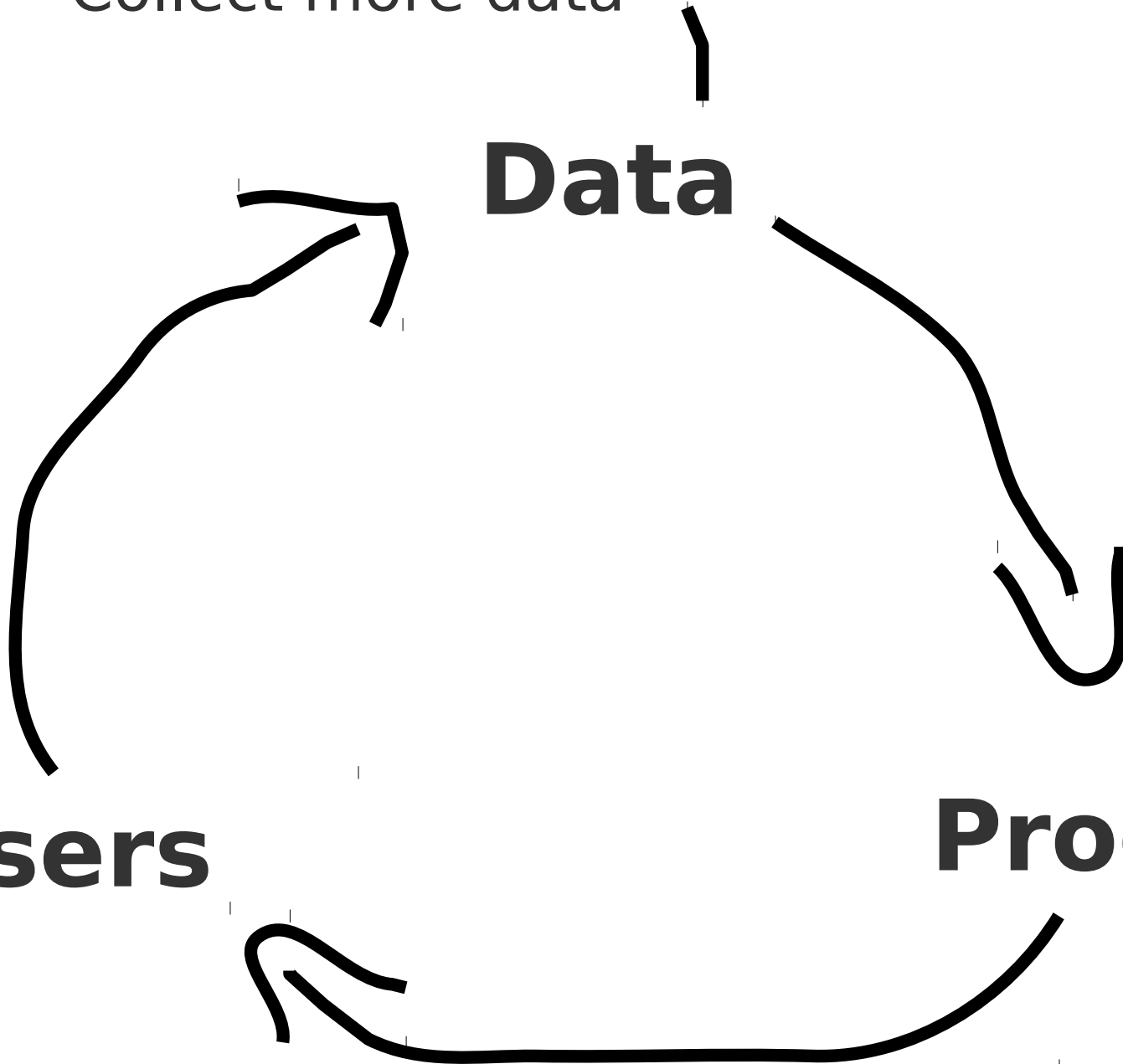


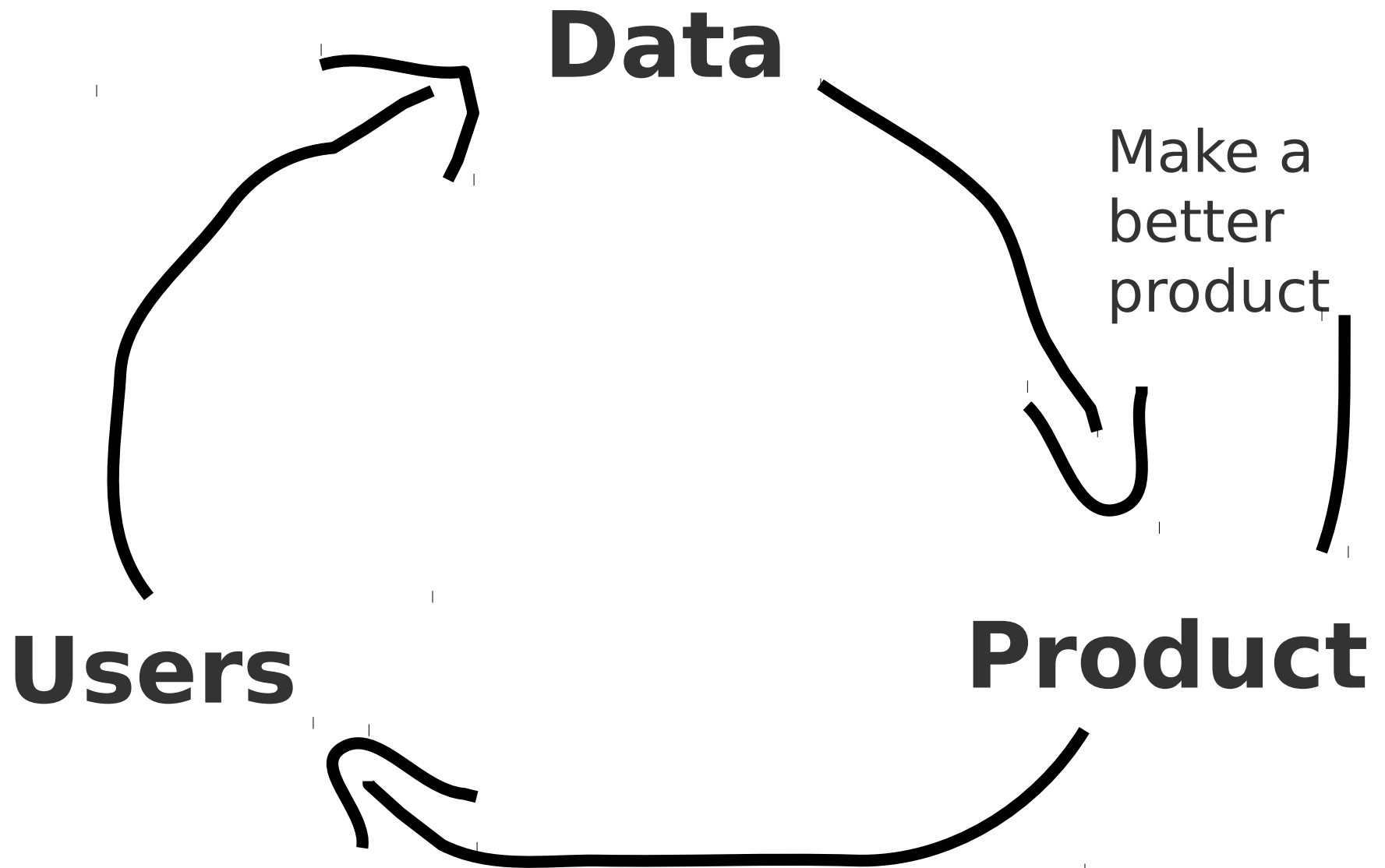
Collect more data

Data

Users

Product



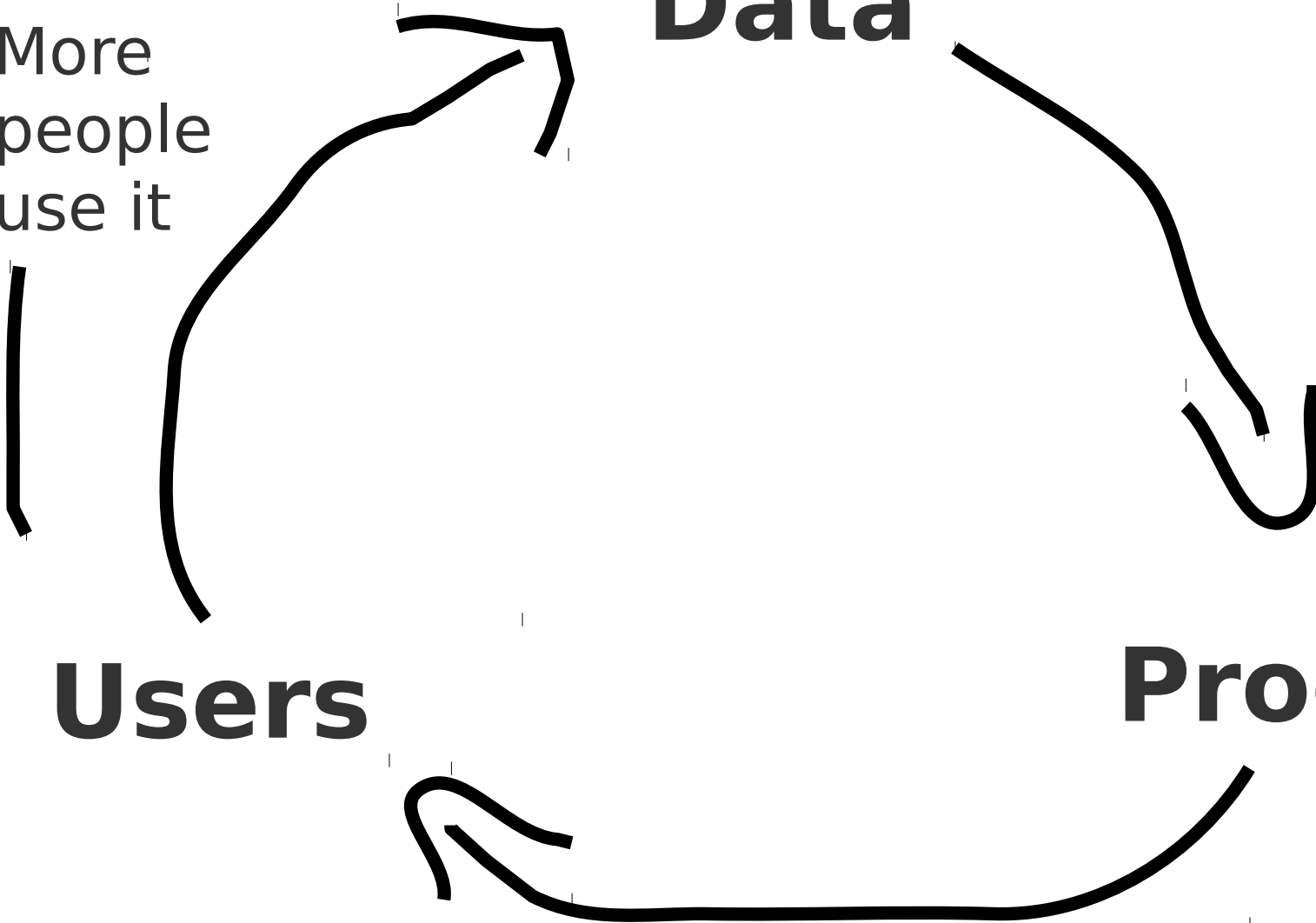


Data

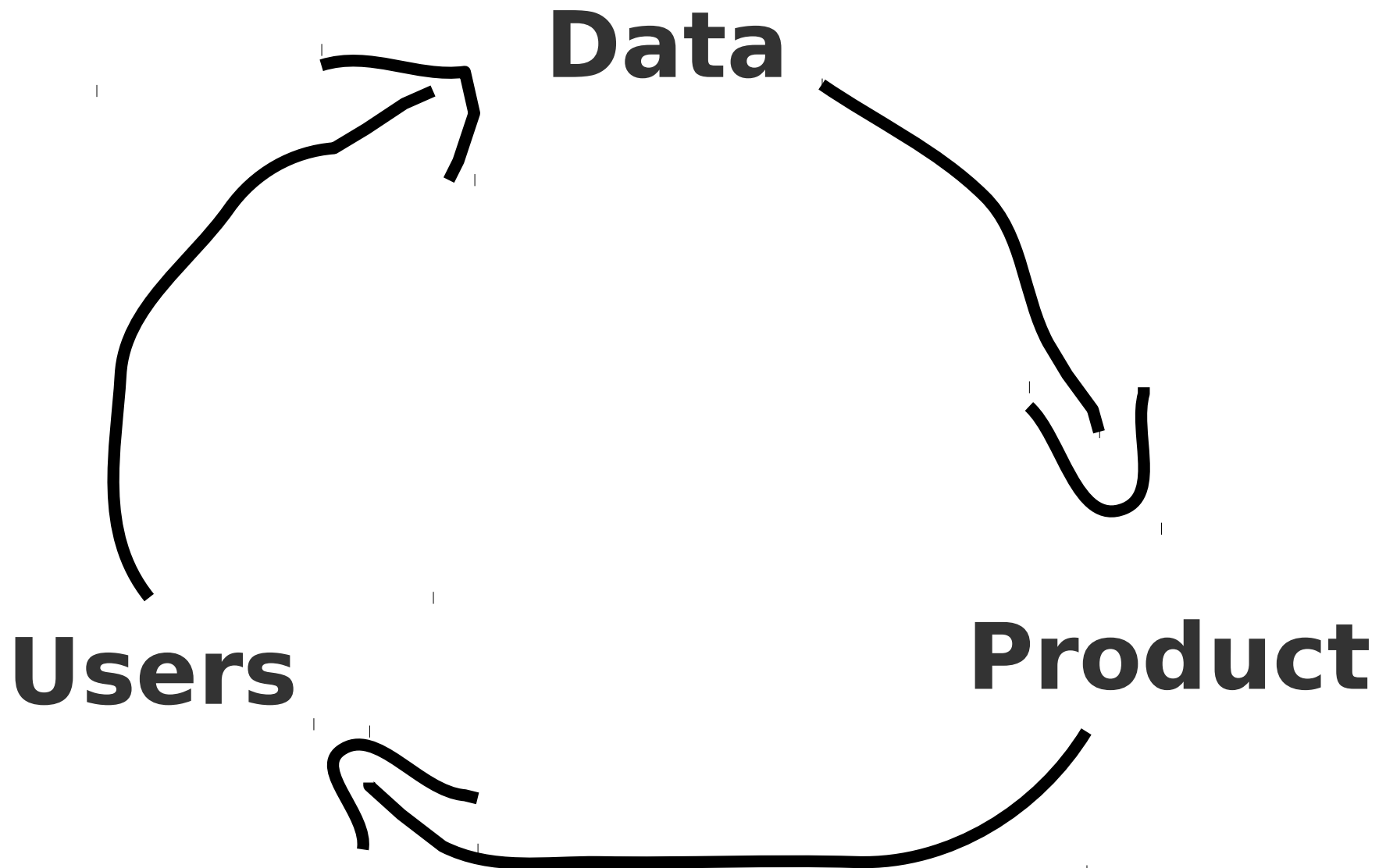
More
people
use it

Users

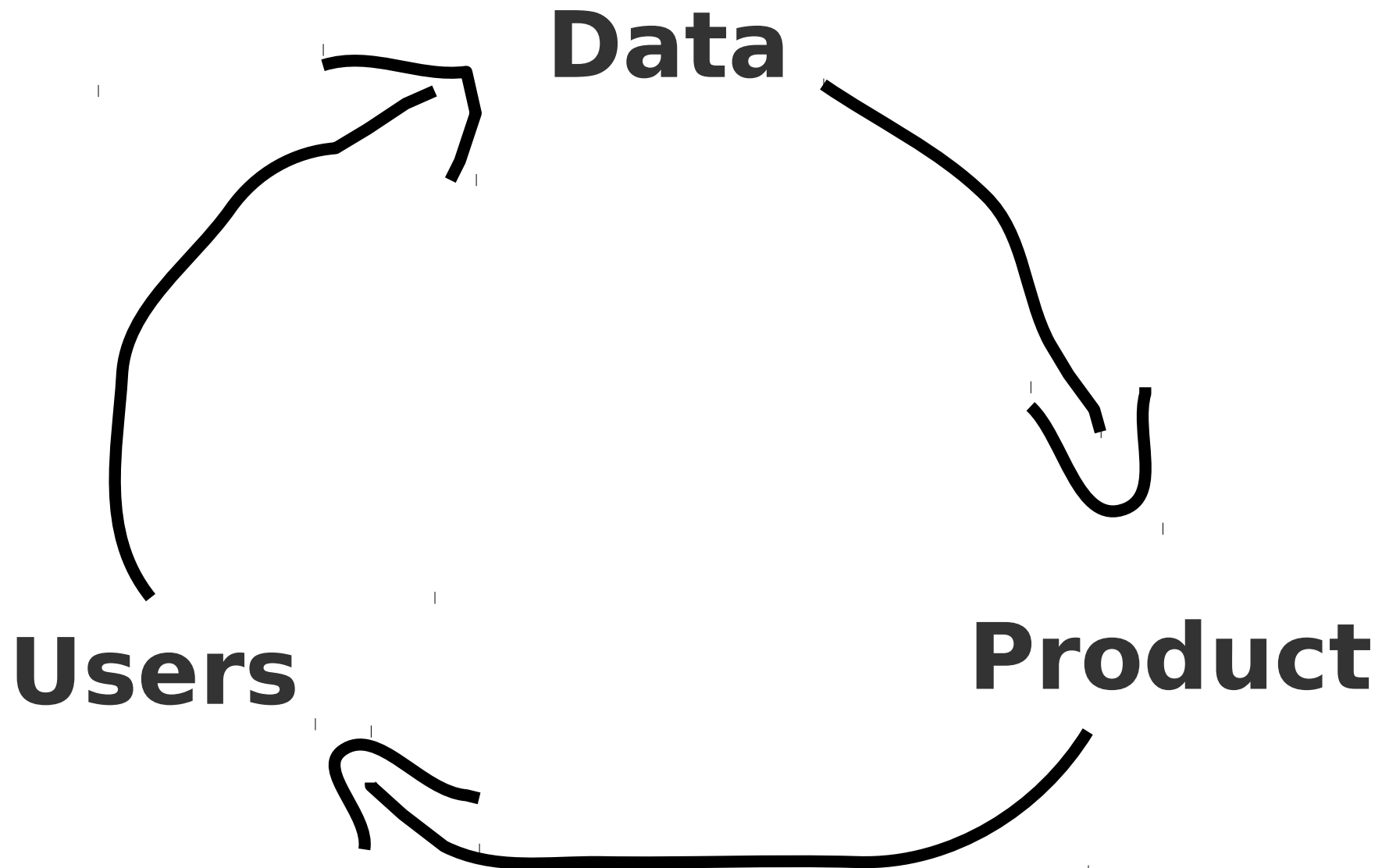
Product



Collect even more data

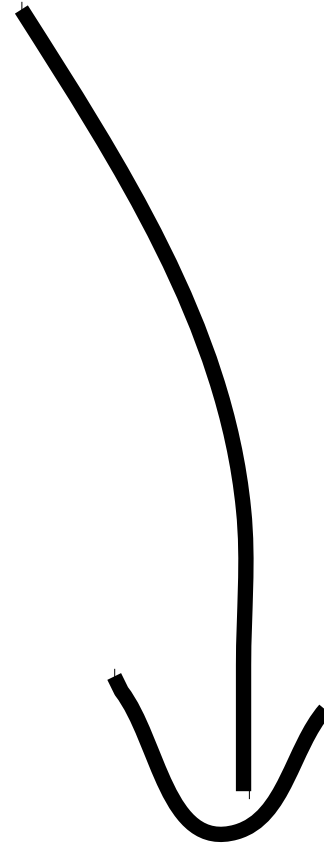


Etc., etc....



This product part will likely have neural network technology involved

Data



Users

Product

Right now, these products are primarily technologies that map $A \rightarrow B$

Product

Right now, these products are primarily technologies that map $A \rightarrow B$

Product

Picture \rightarrow Cat or Dog or Airplane

Right now, these products are primarily technologies that map $A \rightarrow B$

Product

Picture \rightarrow Cat or Dog or Airplane

stock \rightarrow buy or sell

Right now, these products are primarily technologies that map $A \rightarrow B$

Product

Picture \rightarrow Cat or Dog or Airplane

stock \rightarrow buy or sell

Borrower \rightarrow pay back or not pay back loan

Right now, these products are primarily technologies that map $A \rightarrow B$

Product

Picture \rightarrow Cat or Dog or Airplane

stock \rightarrow buy or sell

Borrower \rightarrow pay back or not pay back loan

Speech \rightarrow Text

Right now, these products are primarily technologies that map $A \rightarrow B$

Product

Picture \rightarrow Cat or Dog or Airplane

stock \rightarrow buy or sell

Borrower \rightarrow pay back or not pay back loan

Speech \rightarrow Text

Many many many more examples, as A and B can take many different forms.

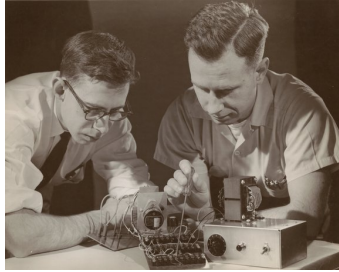
Anything a human can do in about 1 second, a neural network can be trained to do [ish].



Andrew Ng

Famous Fancy AI Person

How did we get here?



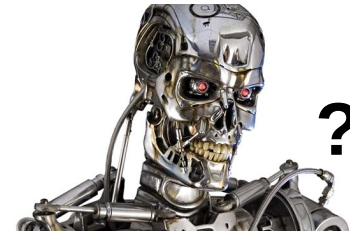
1948:
Hebb's
neuroplasticity
hypothesis,
Hebbian learning

1956:
Rosenblatt
invents the
perceptron

1985:
Rumelhart,
Hinton, and
Williams pioneer
backpropagation
applied in NN
training



2010s:
Deep Mind's
AlphaGo beats
world Go
champion



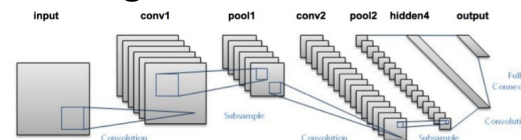
Future

1943:
Warren
McCulloch
and Walter
Pitts define
threshold
logic

1954:
Farley and
Clark simulate
a Hebbian
network with
computers

1965:
Ivankhnenko
and Lappa
publish many
layered
networks

2000s:
Convolutional NN
become state-of-
the-art for object
recognition



Present:
NN software
exists in many
applications



Fine, so neural networks can do some stuff, but if they are not this:



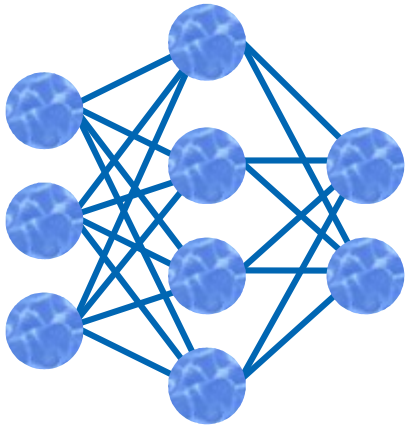
Fine, so neural networks can do some stuff, but if they are not this:



Then what are they?

Neural networks are often explained in graph, math, or code representations.

Graph



Math

$$\begin{matrix} 1 & 4 & 2 \\ 5 & 3 & 1 \\ 8 & 5 & 7 \end{matrix} * \begin{matrix} 9 & 1 & 3 \\ 5 & 3 & 1 \\ 8 & 5 & 7 \end{matrix}$$

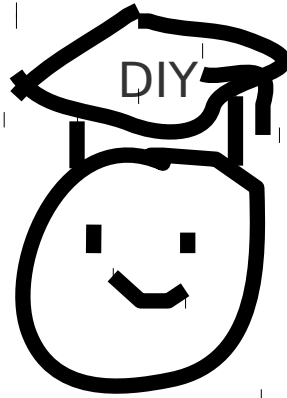
Code

```
Data = read.csv('/home/data.csv')
X = data[:,1:10]
y = data[:,0]

model = graph()
model.input(3)
model.connected(4)
model.connected(2)
model.softmax()
model.compile()

model.train(X, y)
```


As someone with a mix of formal and self-taught training,

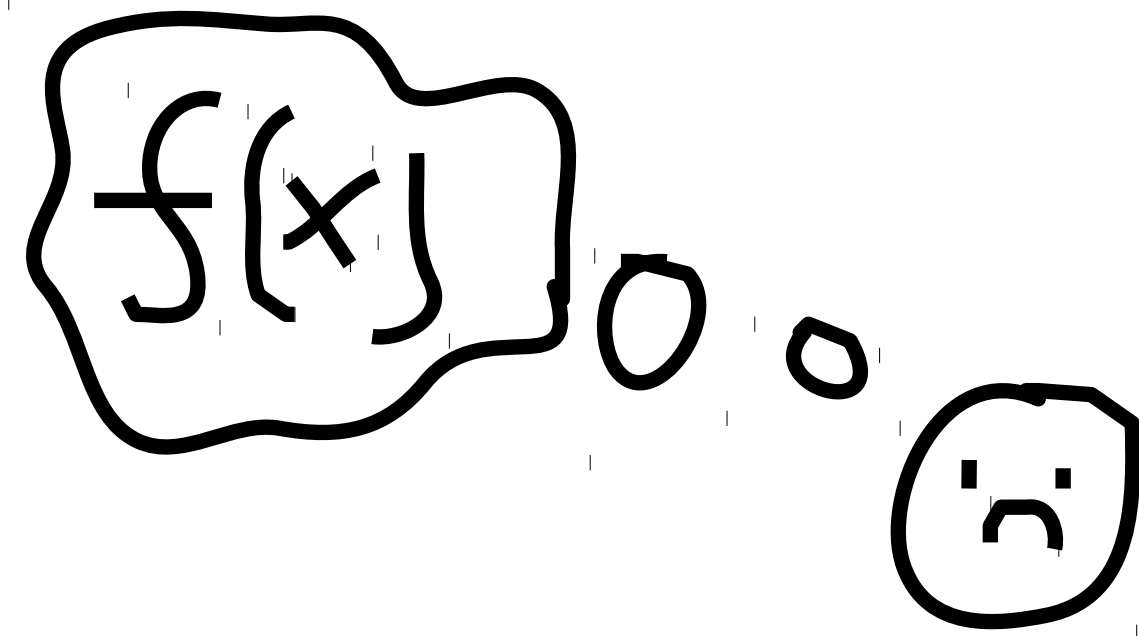


I've found learning each of these representations and how they compliment each other gives a deep understanding you can use to **build solutions now**,

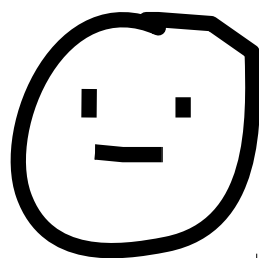
and

a platform to **continuously learn more** with the plethora of information that is freely available.

This may be intimidating at first. You may feel like you need some fancy degree or years of training to understand the math.

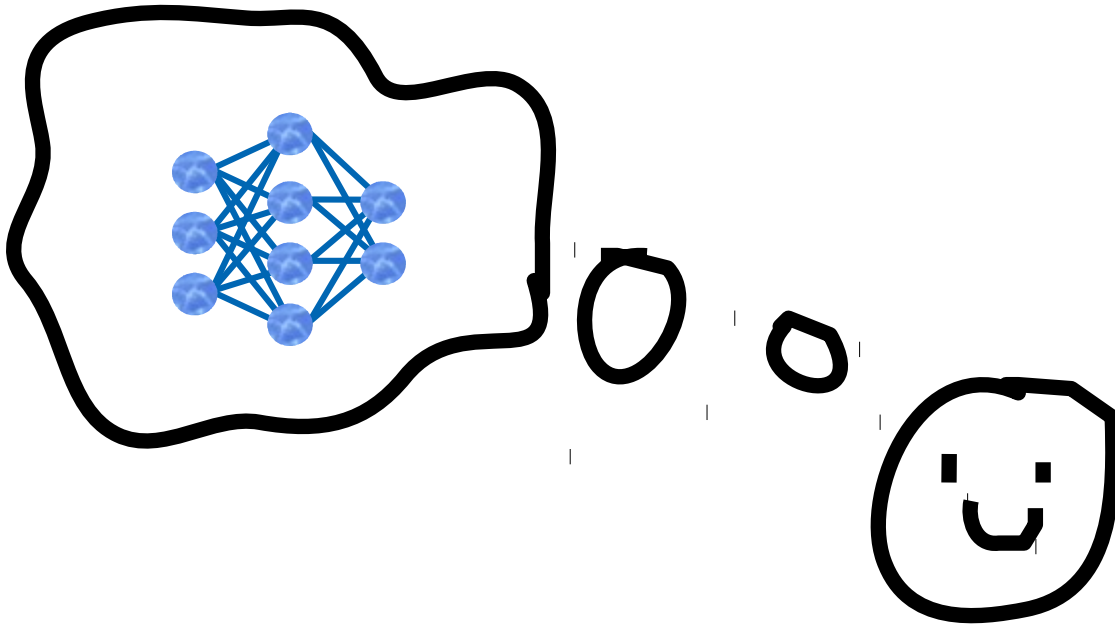


Don't feel like there are smart people who are capable of figuring this out, and dumb people who aren't.

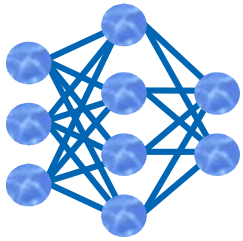


Anyone can learn the math behind neural networks. Even “fancy” people get bored and take short-cuts.

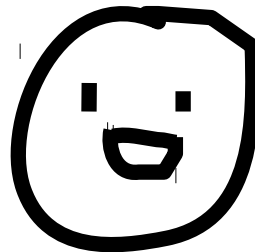
I was trained in industrial process engineering. So when I started learning, I liked the graph representation best to get a mental picture of what's going on.



I was trained in process design engineering, so I liked the graph representation best to get a mental picture of what's going on.

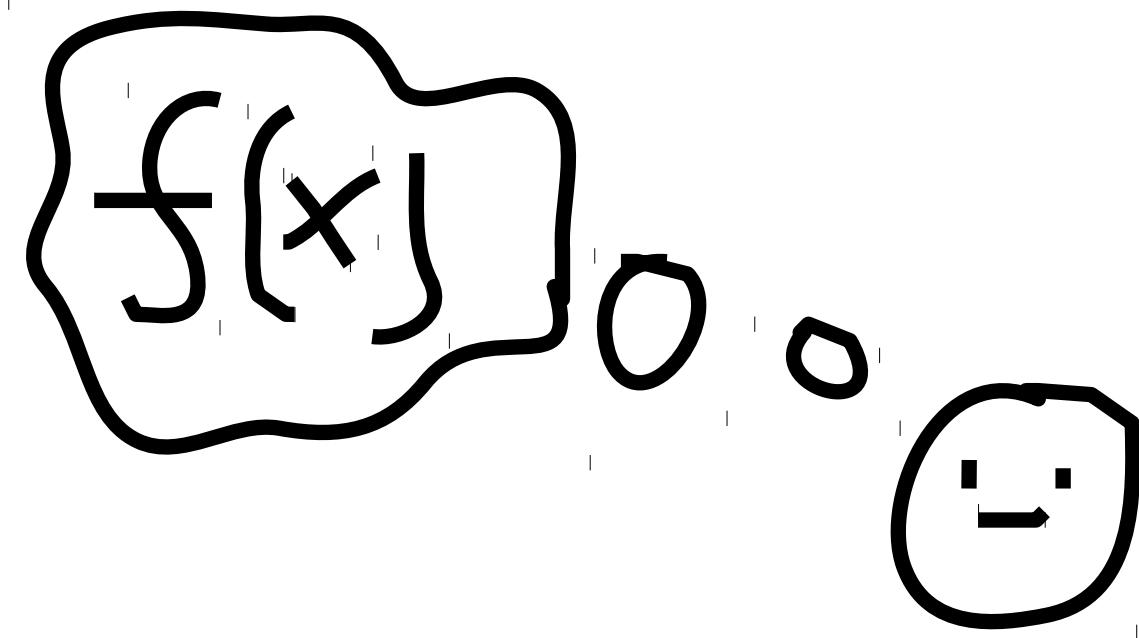


```
model = graph()  
model.input(3)  
model.connected(4)  
model.connected(2)  
model.softmax()  
model.compile()
```

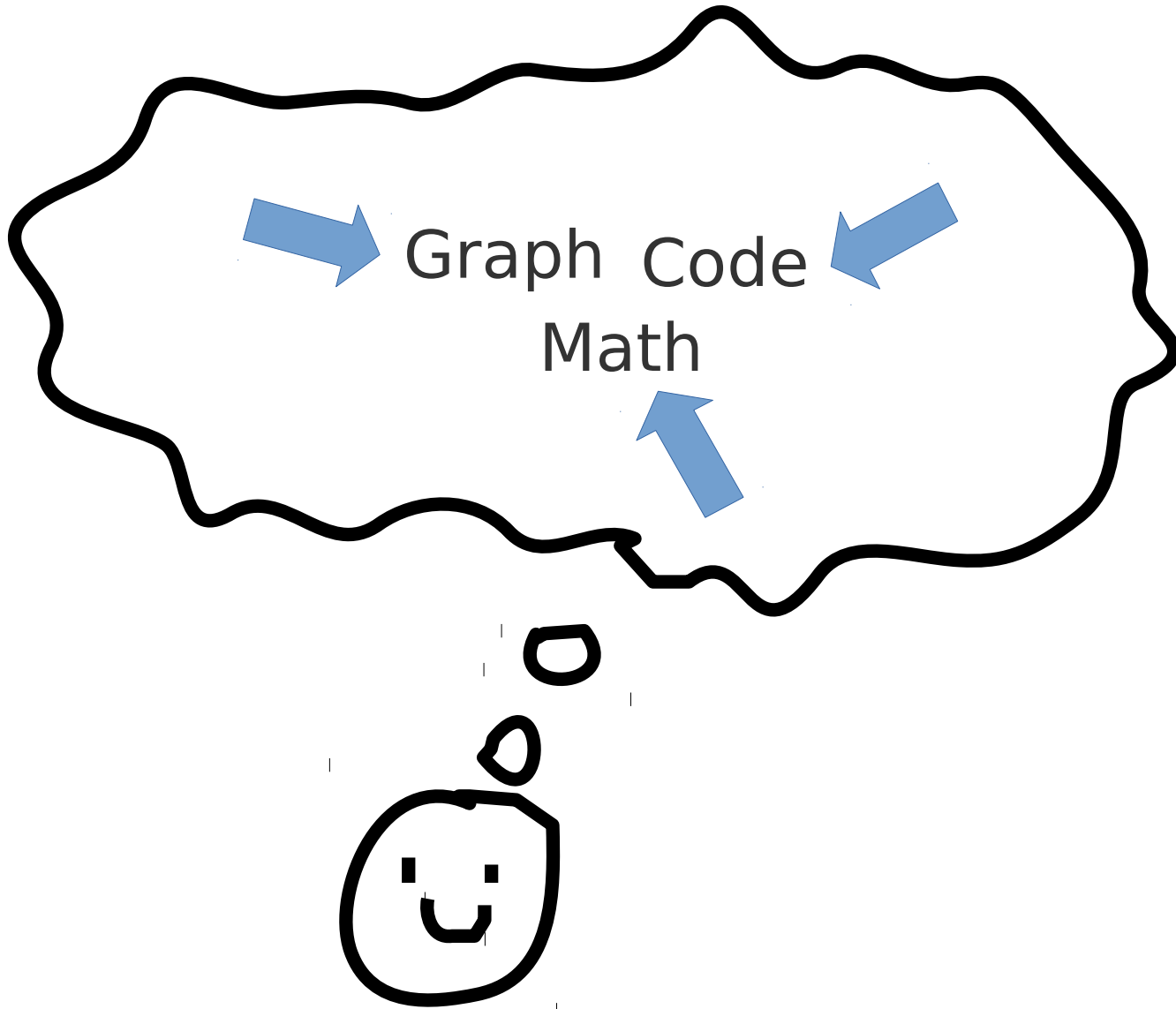


But I also want to make things that actually work *now*, without waiting around for everything to be perfect. So, I would think about these models through the code representation too.

When starting a new project, or refining a current one, I would drop the graph and code for a minute to go through the raw math.



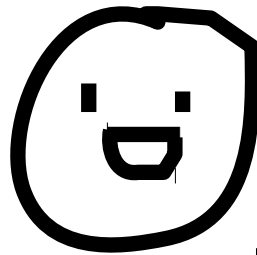
After a while, these three representations began to unify in my brain and I started to feel like I knew what was going on.



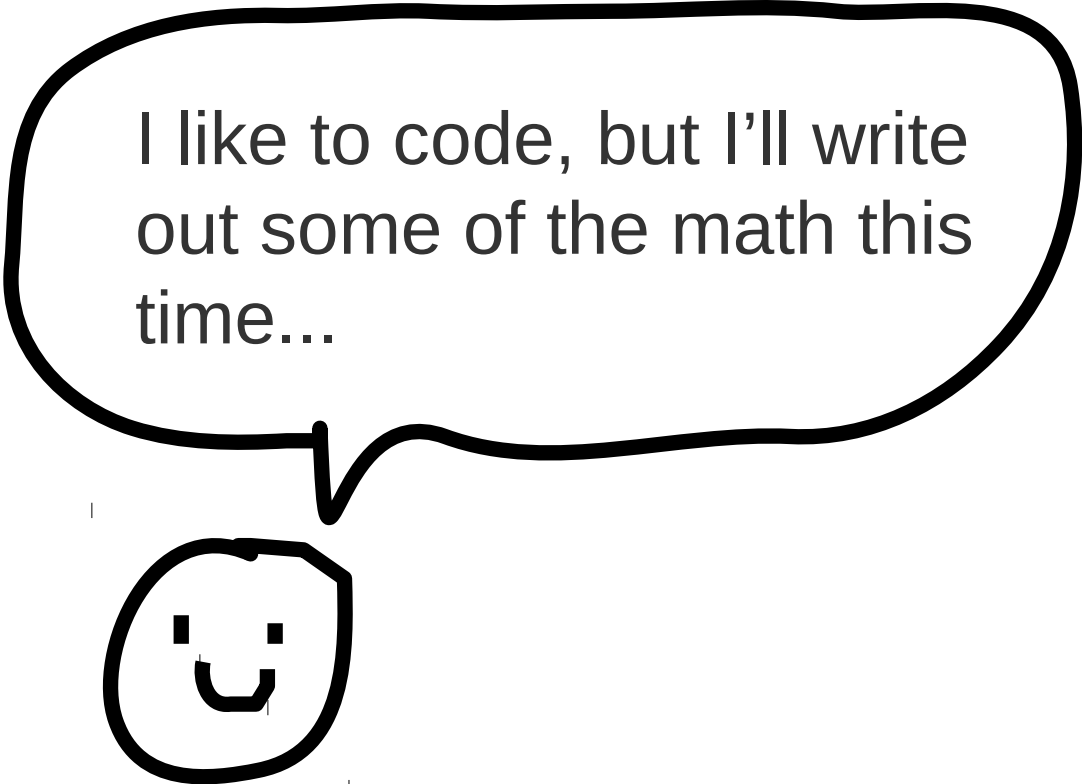
Understanding these types of models and applying them to real-world problems lead to a lot of great things. There was a lot of time spent and tons of failures, but I couldn't believe how well some of my projects went.

\$\$\$

You Win!



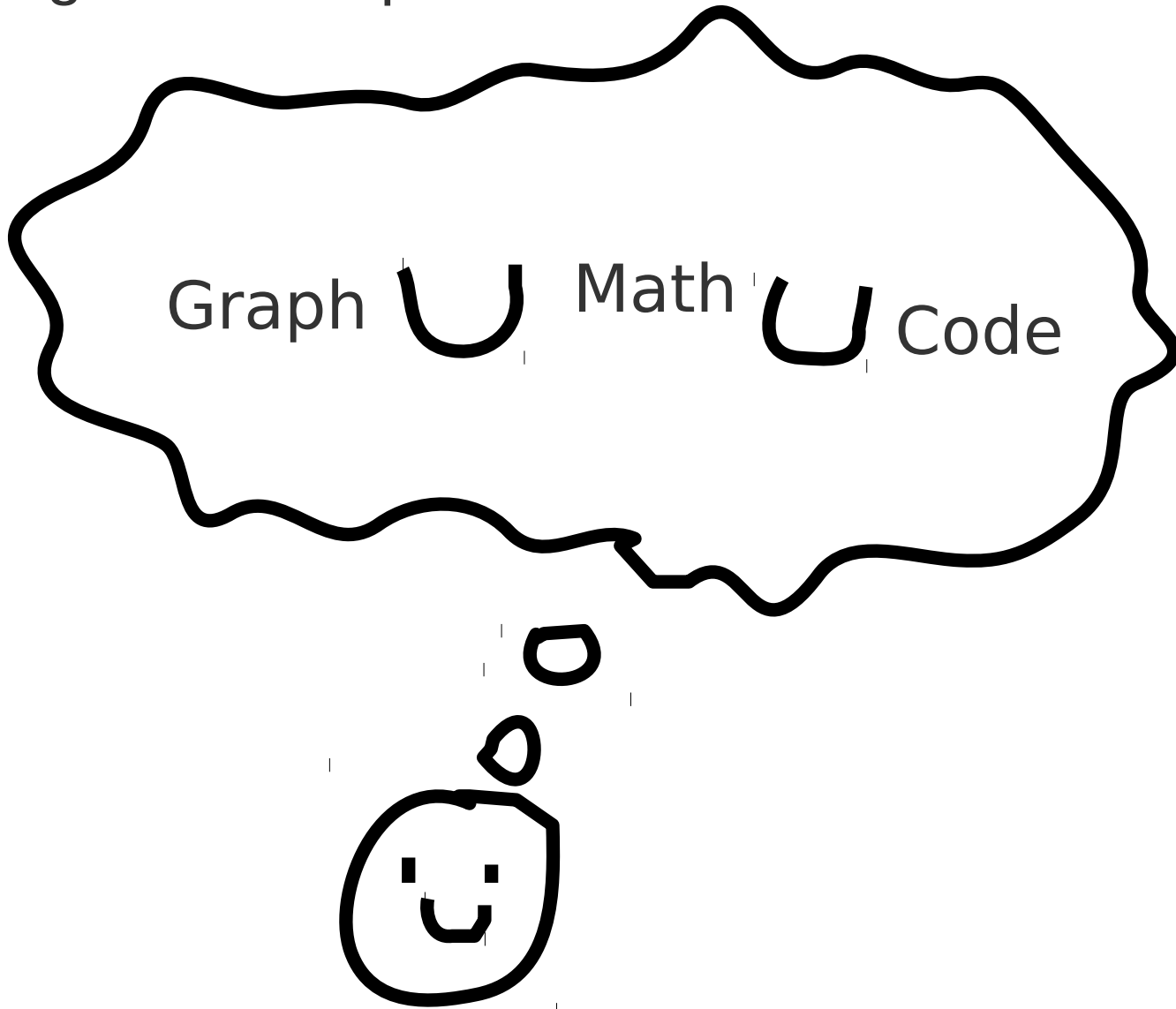
While you learn, play to your strengths to make **fast and furious progress**,



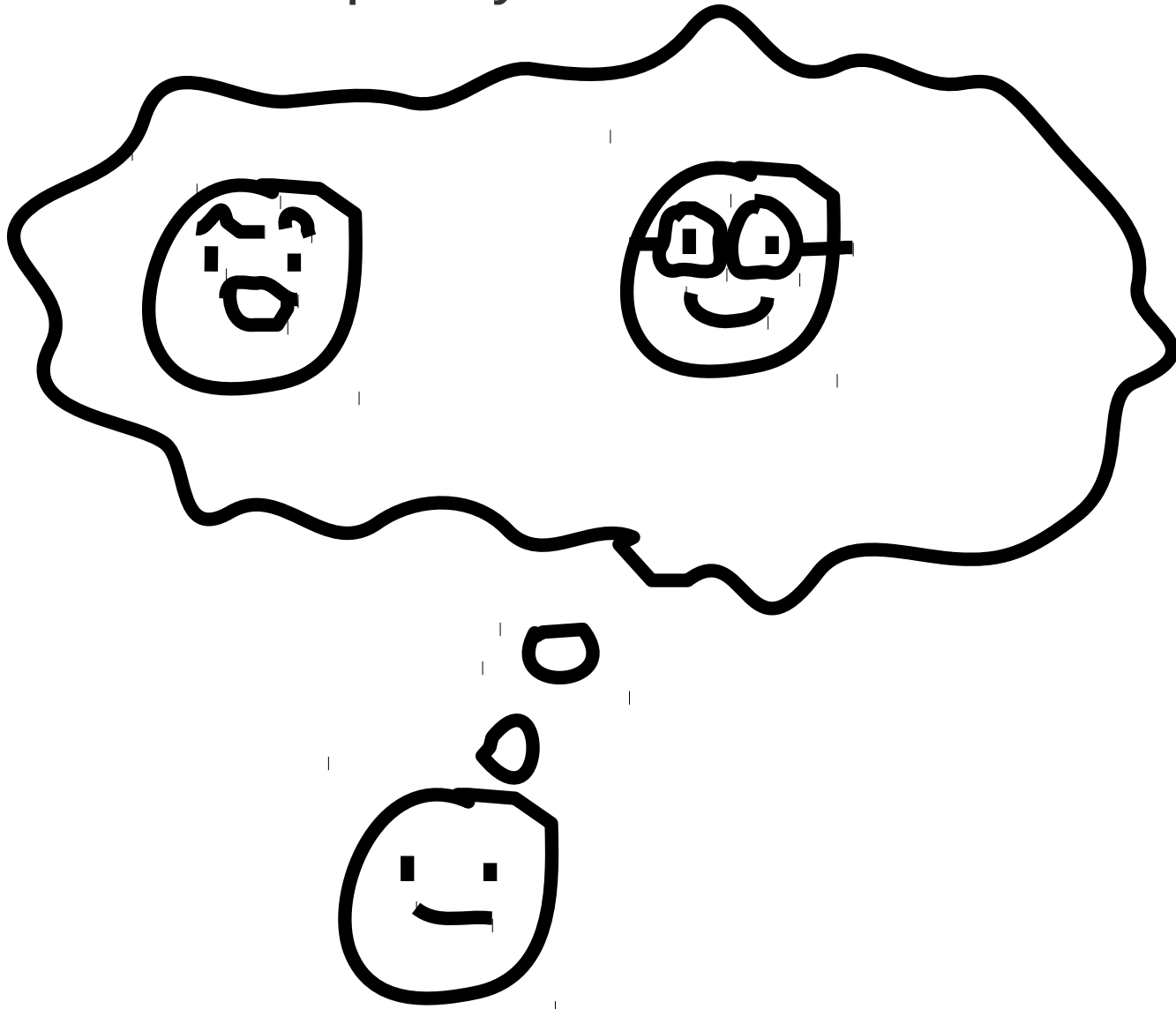
I like to code, but I'll write out some of the math this time...

but also periodically look back and challenge yourself with other representations

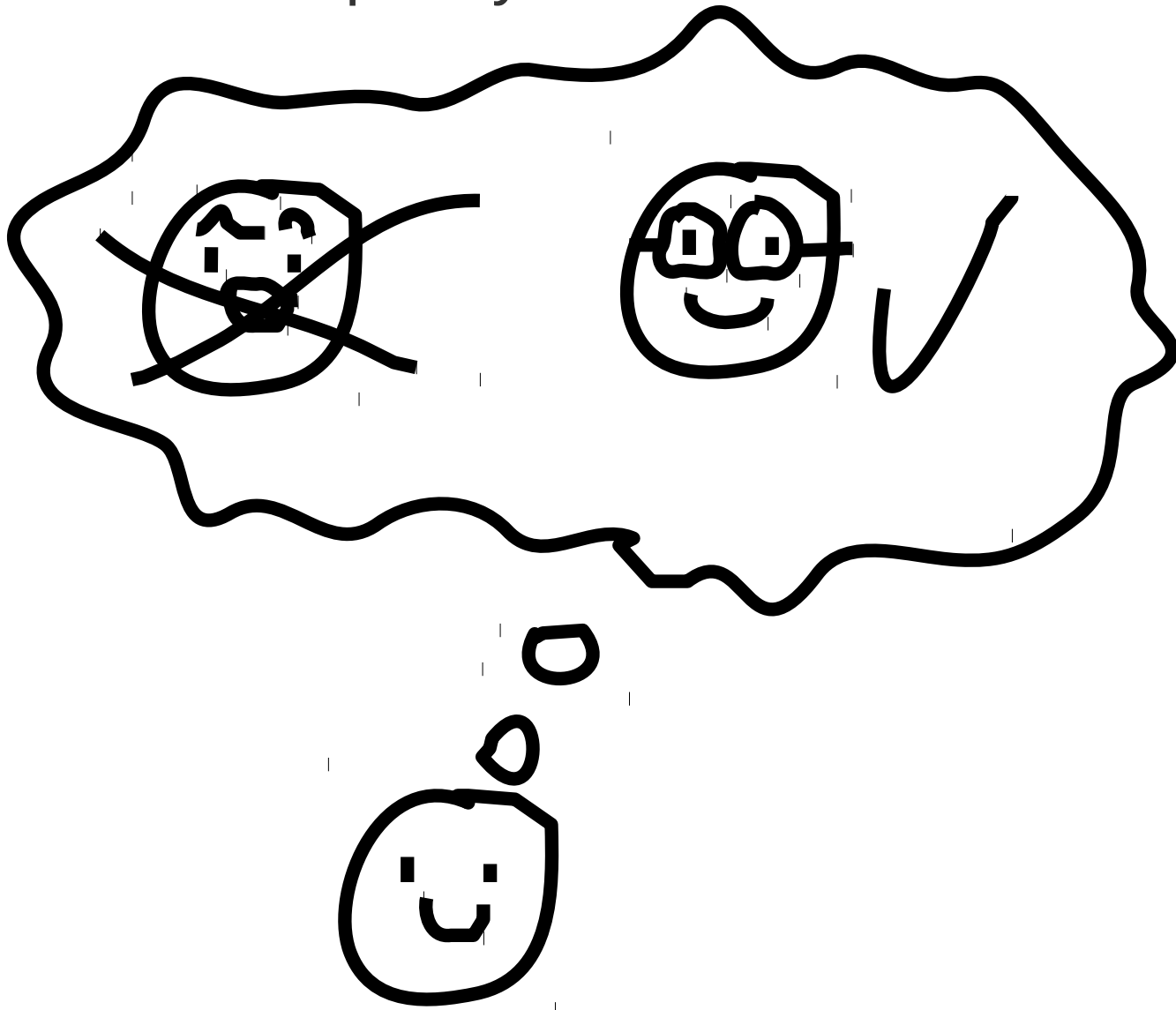
Mentally unify these three representations, and you will start feeling like an expert



Never become arrogant, and constantly learn more, or you what you know will quickly become obsolete.



Never become arrogant, and constantly learn more, or you what you know will quickly become obsolete.



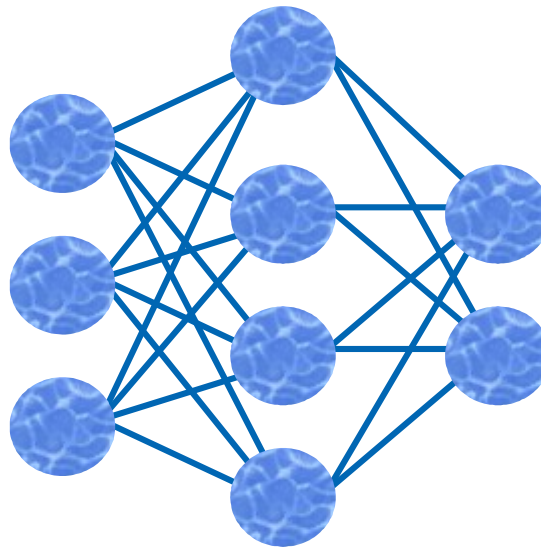
So neural networks are easy?!?!

Well, I hope to make them as simple as possible here.

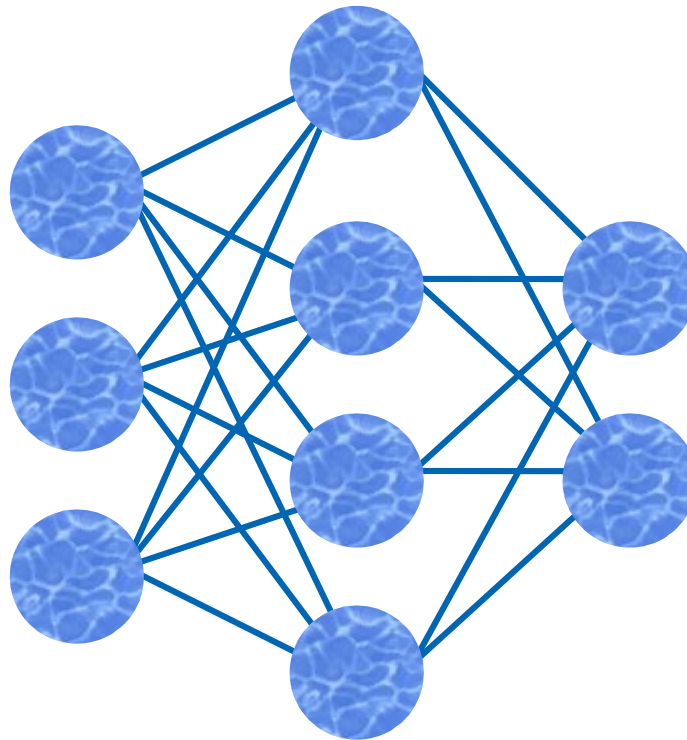
We will walk through an example primarily using the graph representation, but with elements of the math and code representations as well.

Time to get weird...

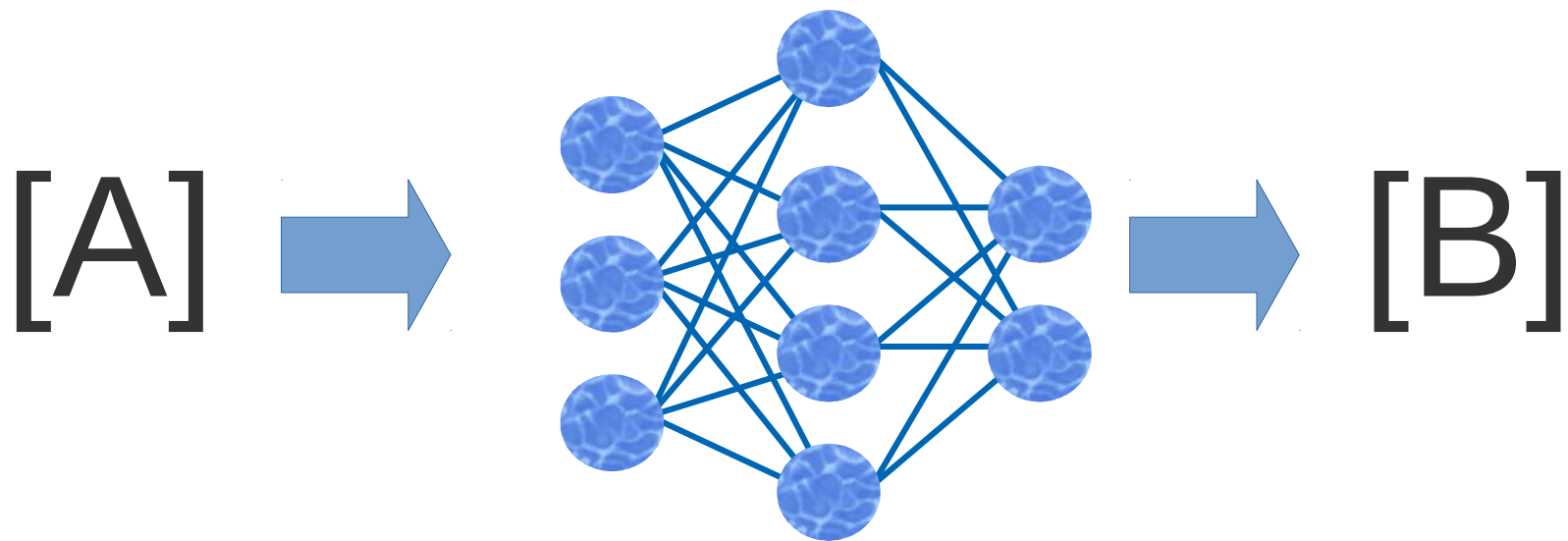
You saw this before as the graph representation of a neural network:



In the graph representation, it appears that neural networks are made of lines and dots



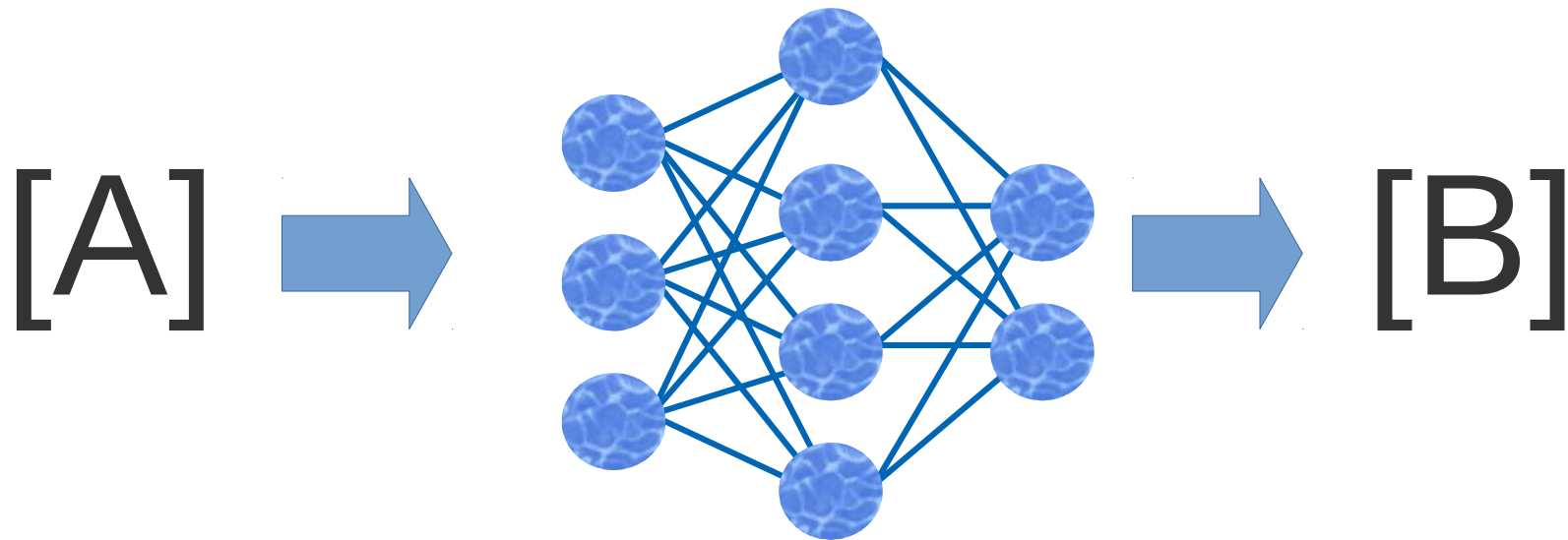
Remember, for now most neural net technologies map an “A” to a “B”



What is an “A” and “B”

Remember the examples? Lets say you work at a bank, and you want to know who to award loans to.

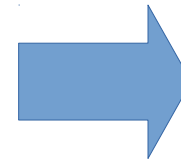
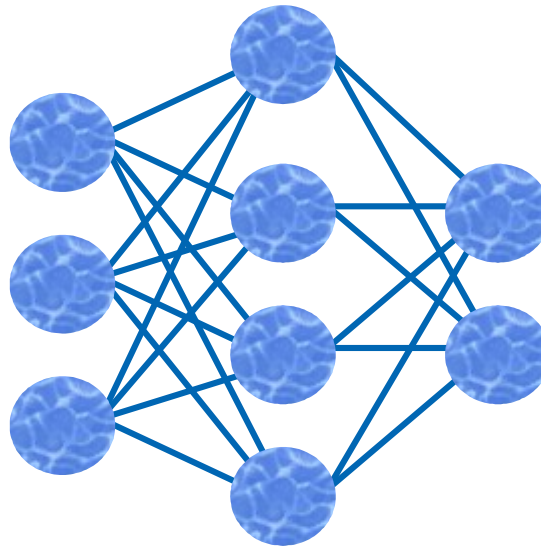
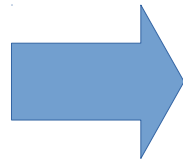
Borrower → pay back or not pay back loan



Borrower

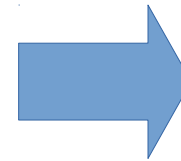
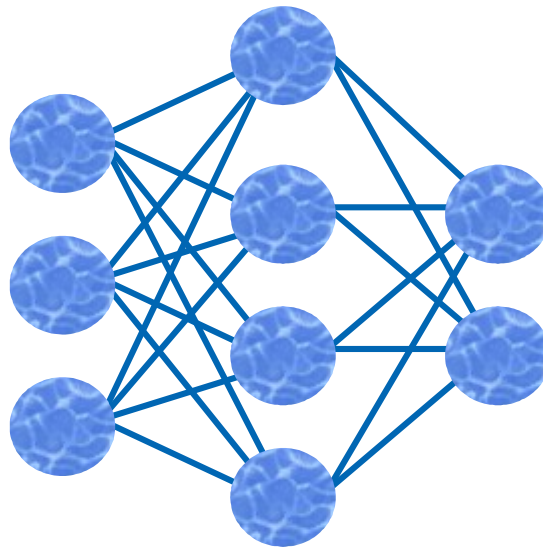
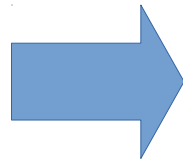
pay back or
not pay back
loan

[A]



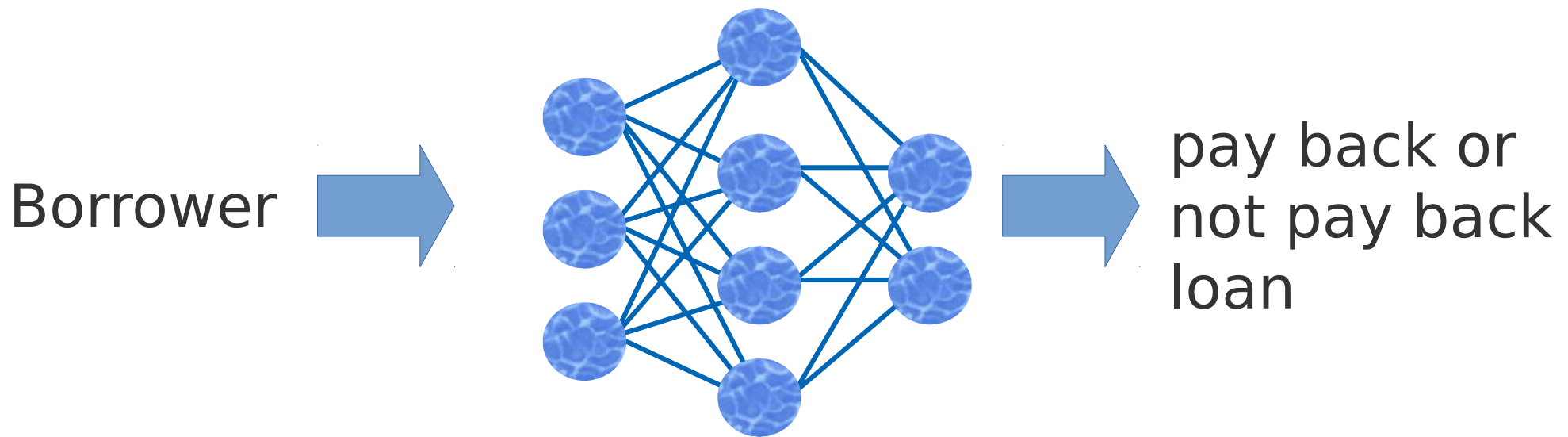
[B]

Borrower



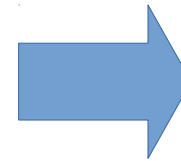
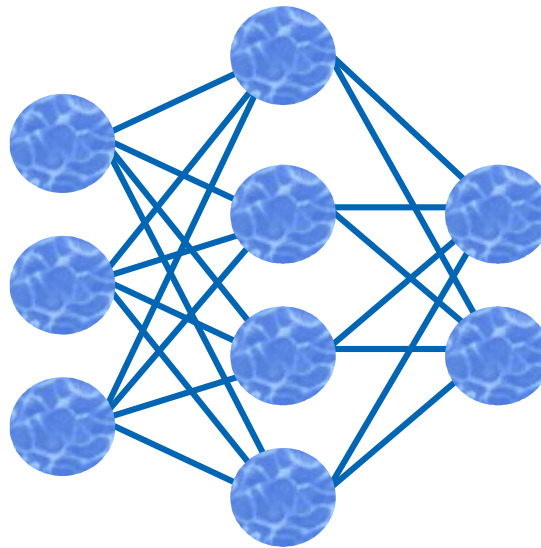
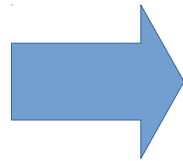
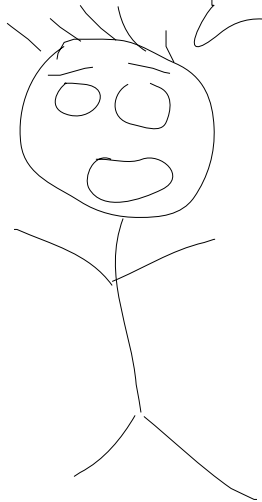
pay back or
not pay back
loan

Easy. so we toss a loan applicant in there and see what comes out.



Easy. so we toss a loan applicant in there and see what comes out..

What are you
Doing?!?!?



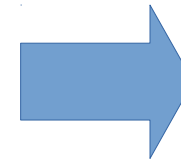
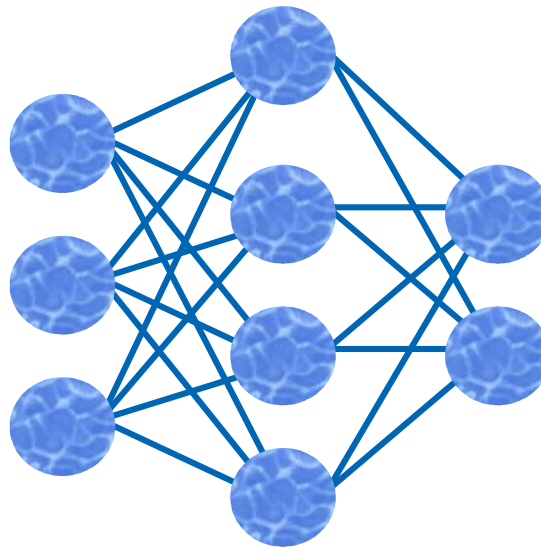
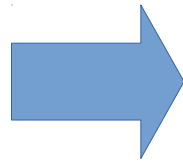
???

Of course not. We put data in. A data set describing loan applicants and their probability to pay back their next loan

Number of
Loans Paid
Back

Number of
Active
Loans

Times
moved in
last 10
years



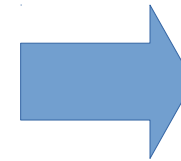
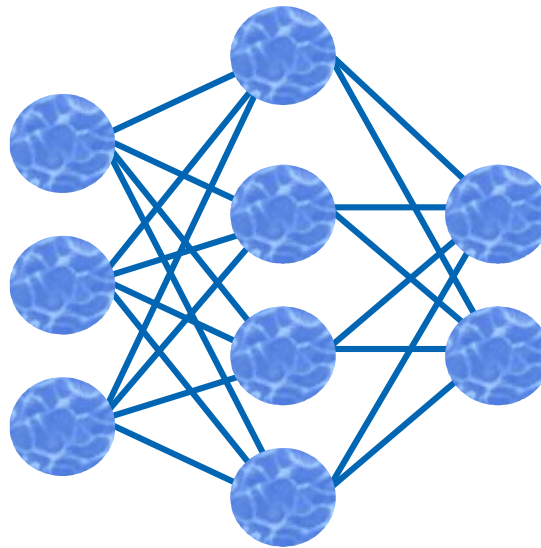
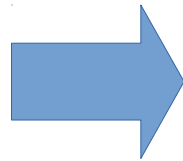
Probability
to pay back
and not pay
back

But how does the neural net know anything?

Number of
Loans Paid
Back

Number of
Active
Loans

Times
moved in
last 10
years



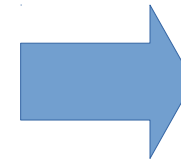
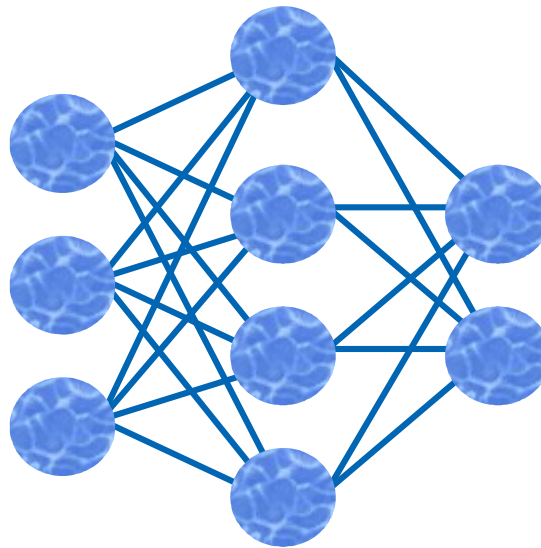
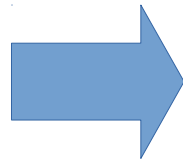
Probability
to pay back
and not pay
back

It has weights, which are numbers in the network that help make the output from an input

Number of
Loans Paid
Back

Number of
Active
Loans

Times
moved in
last 10
years



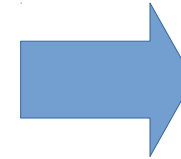
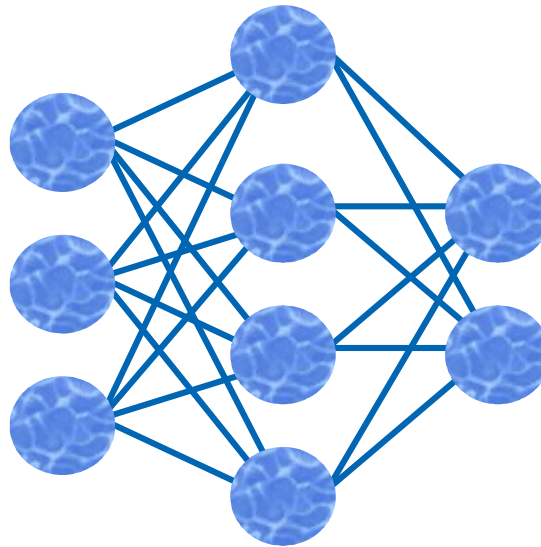
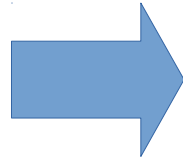
Probability
to pay back
and not pay
back

These weights have to be really “smart” or the network will give dumb results

Number of
Loans Paid
Back

Number of
Active
Loans

Times
moved in
last 10
years



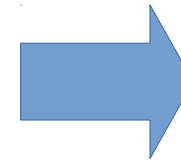
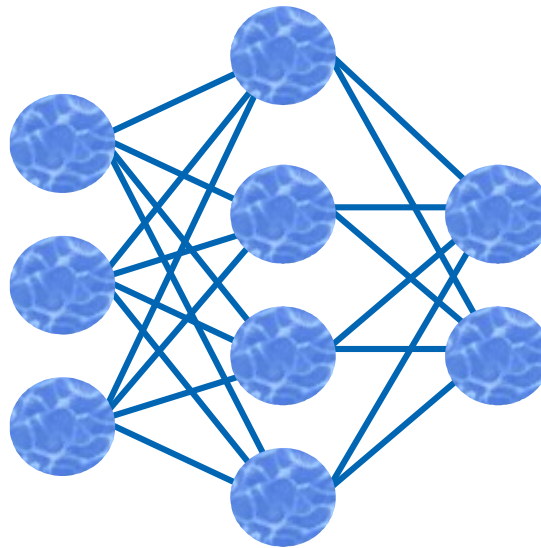
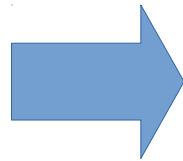
Probability
to pay back
and not pay
back

How do we get smart weights?

Number of
Loans Paid
Back

Number of
Active
Loans

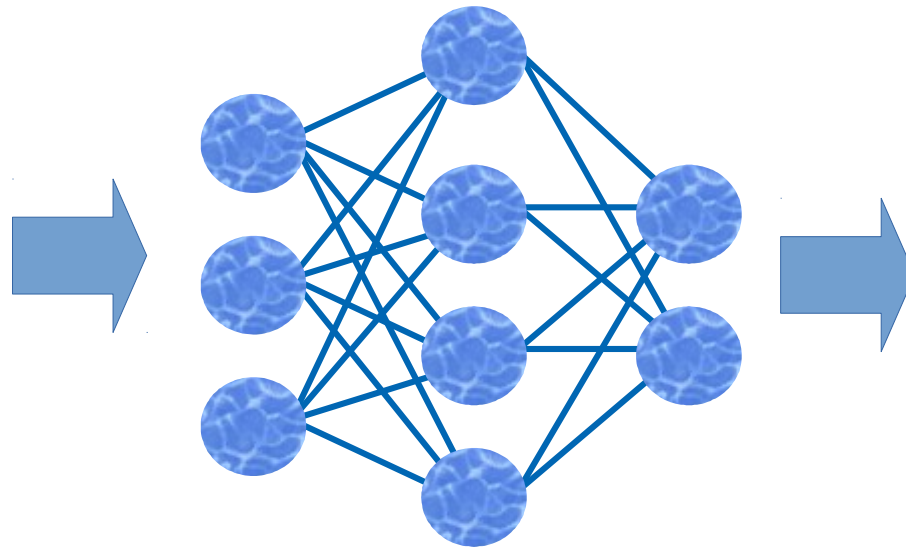
Times
moved in
last 10
years



Probability
to pay back
and not pay
back

Well, we use data from more than just one person, we use data from many people

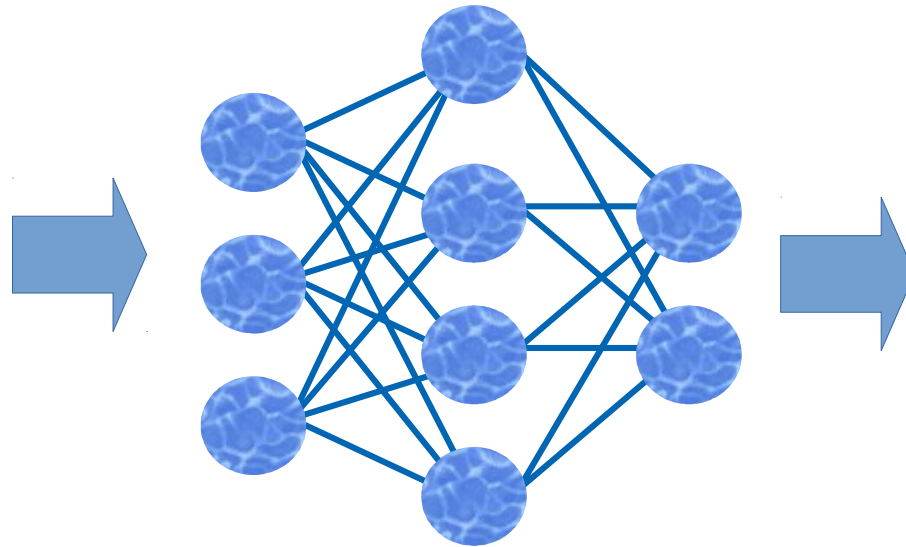
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1

And carefully change the weights
iteratively through a process called...

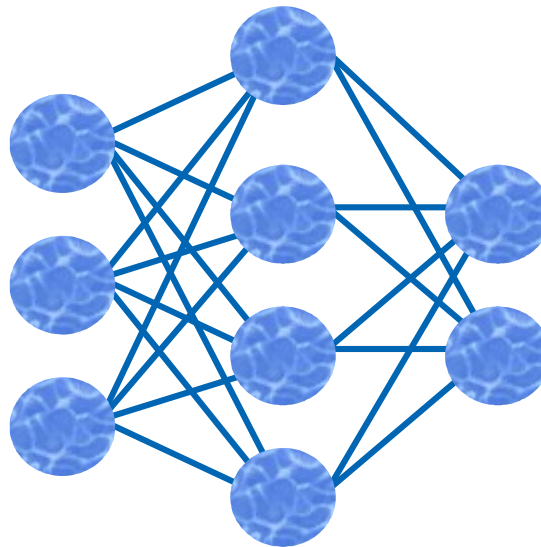
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1

Training!!

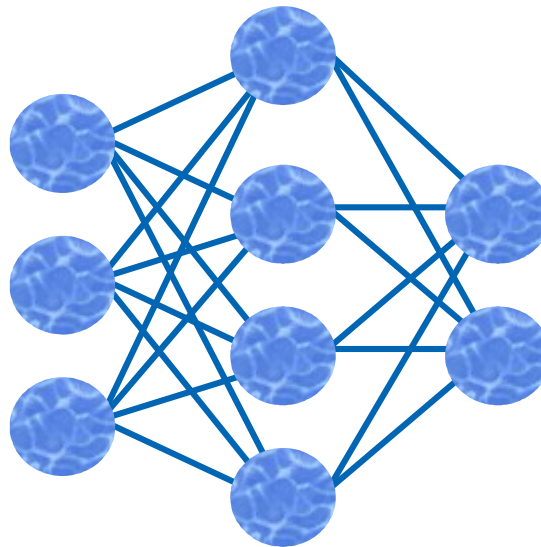
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1

We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

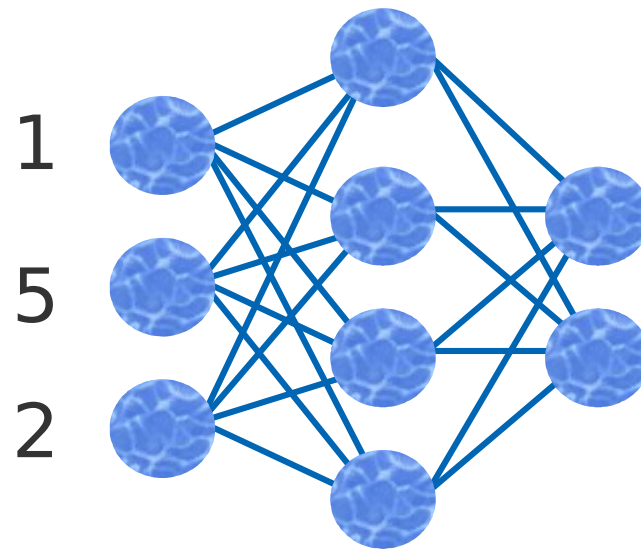


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

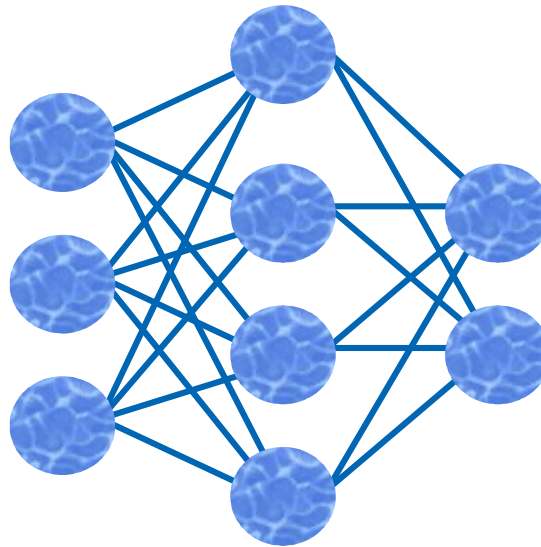


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

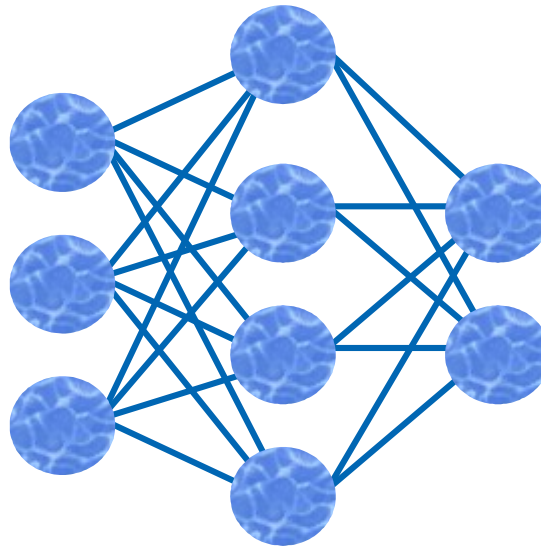


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

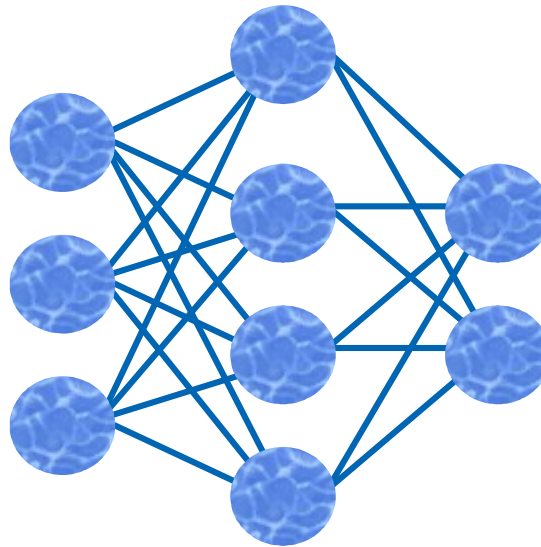


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

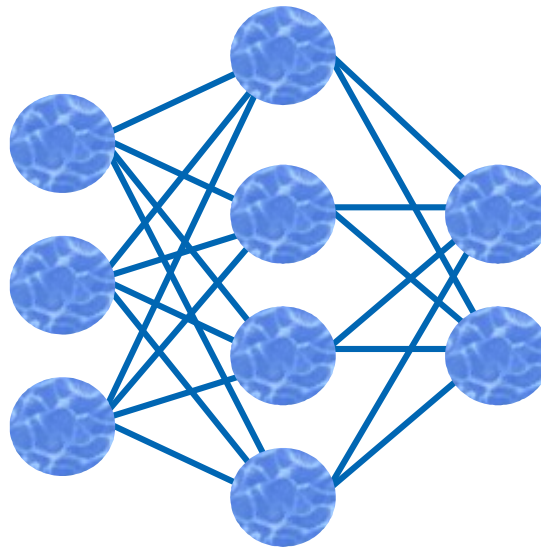


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

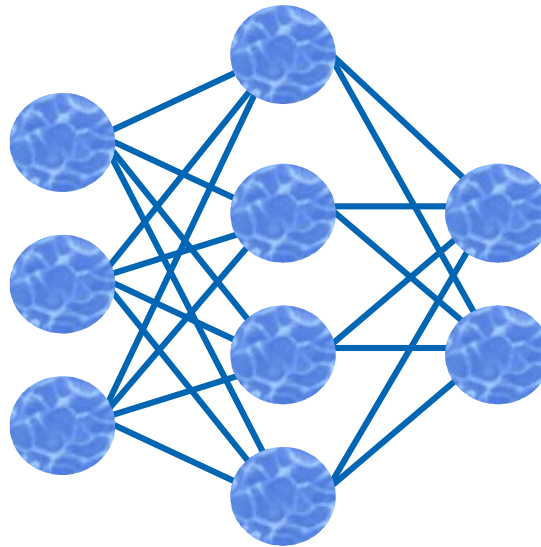


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed an observation into the network

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

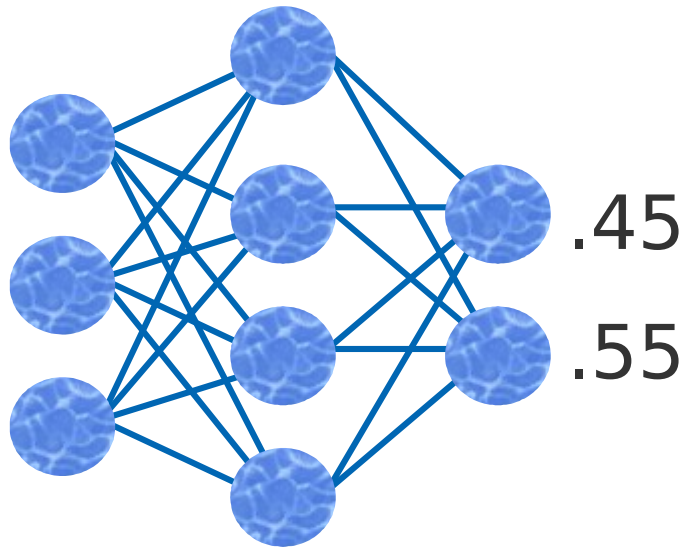


ID	P1	P0
001	0	1
002	1	0
003	0	1



Get an output

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

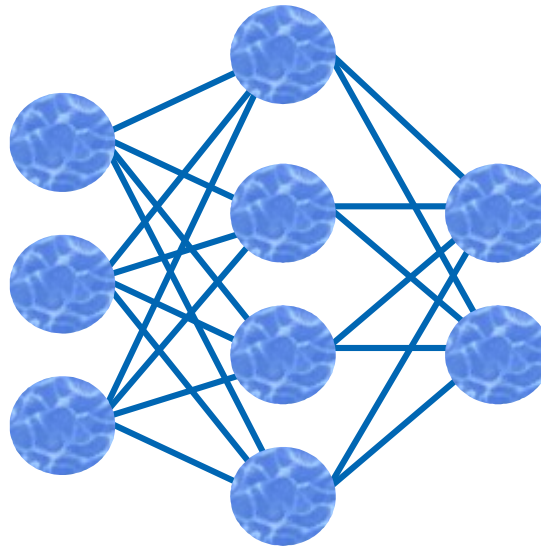


ID	P1	P0
001	0	1
002	1	0
003	0	1



Calculate the error compared to the “B” or labels

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



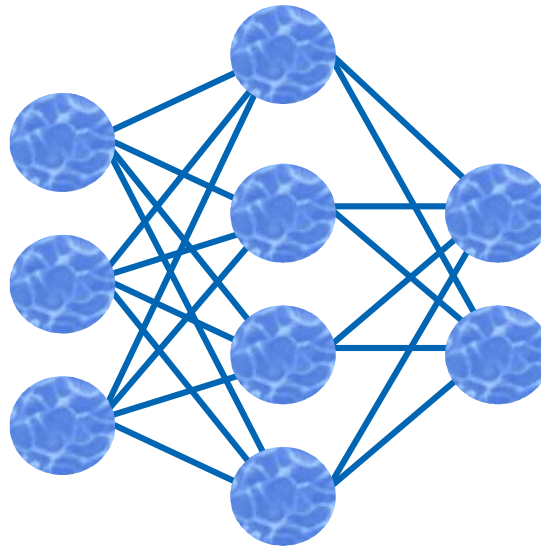
E
r
r
o
r

ID	P1	P0
001	0	1
002	1	0
003	0	1



Then, using the error, go backwards and carefully change the weights in the network to get closer to the labeled values

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

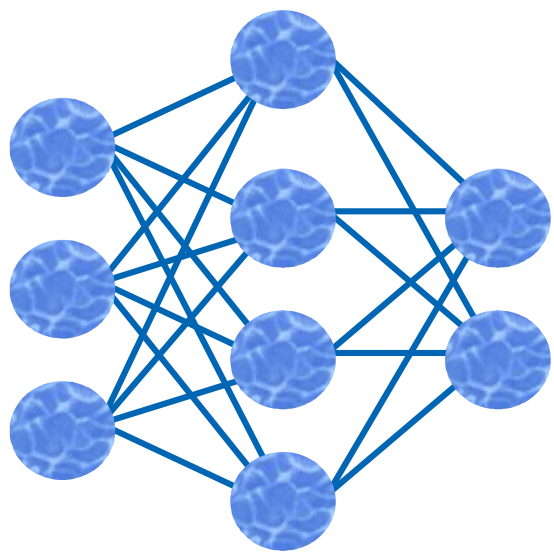


E
r
r
o
r

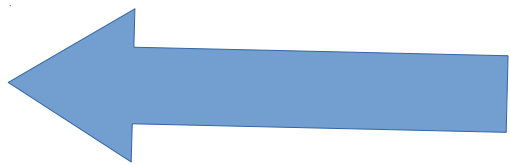
ID	P1	P0
001	0	1
002	1	0
003	0	1



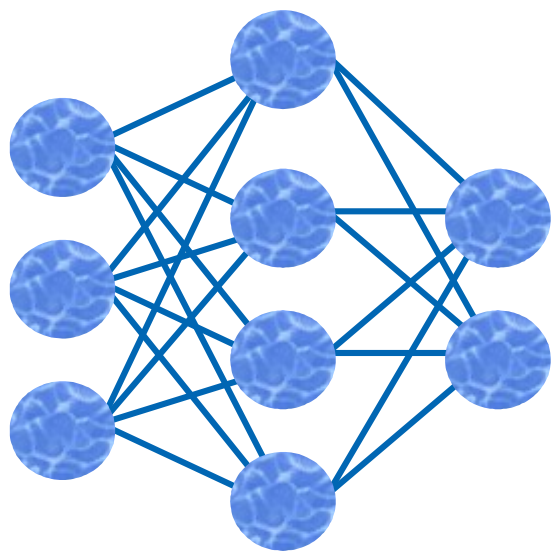
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



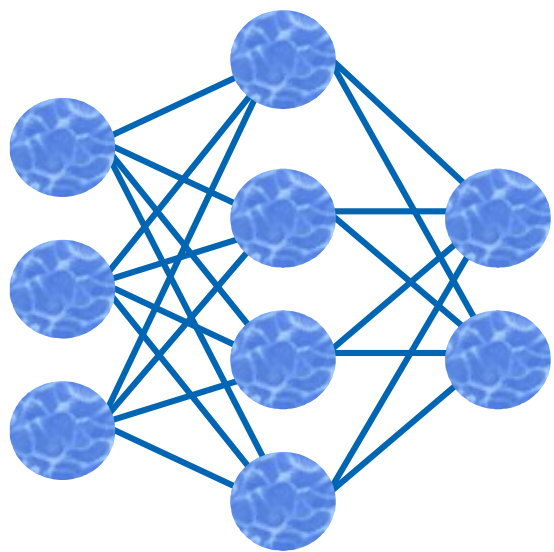
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



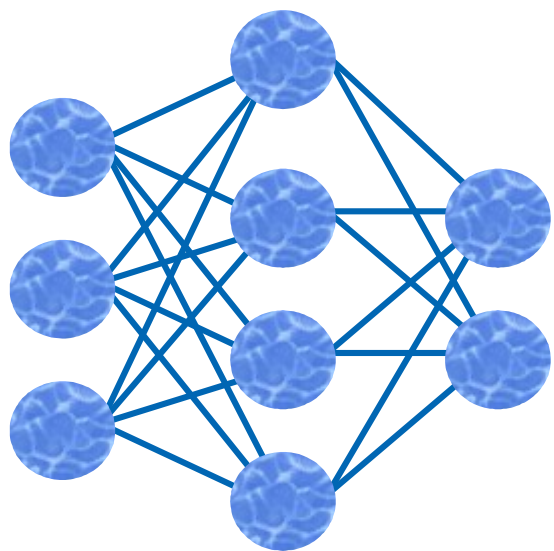
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



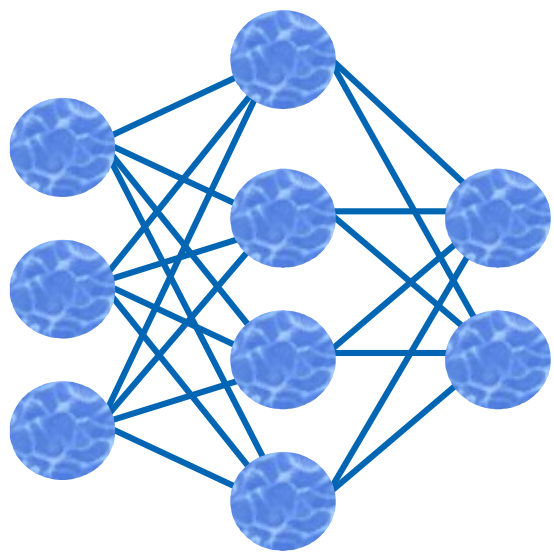
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

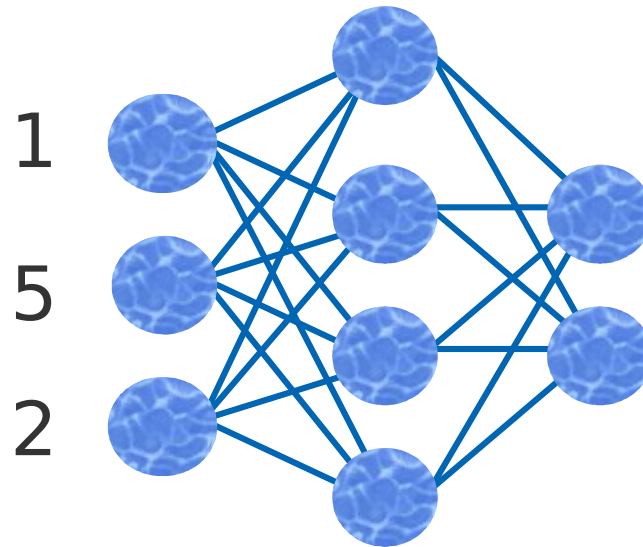


ID	P1	P0
001	0	1
002	1	0
003	0	1



After going completely backwards,

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

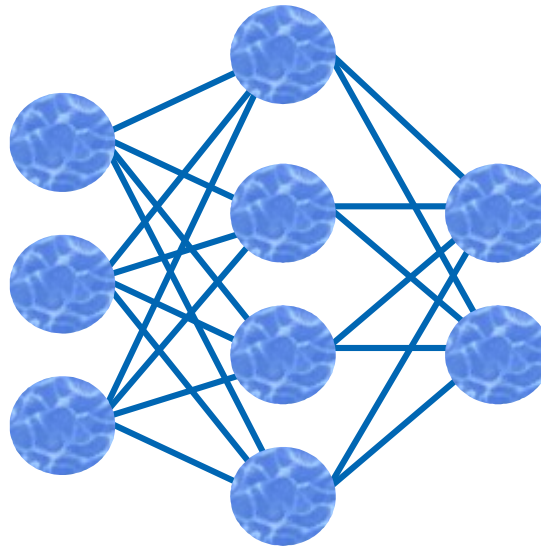


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed another observation into the network, and go forwards.

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

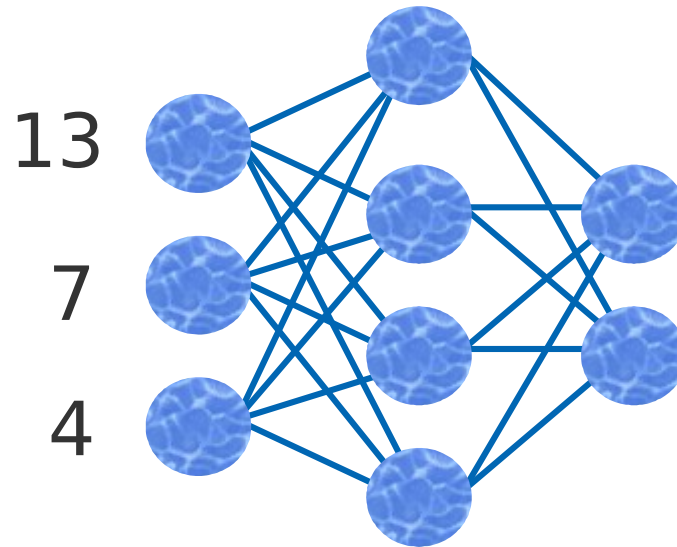


ID	P1	P0
001	0	1
002	1	0
003	0	1



We feed another observation into the network, and go forwards.

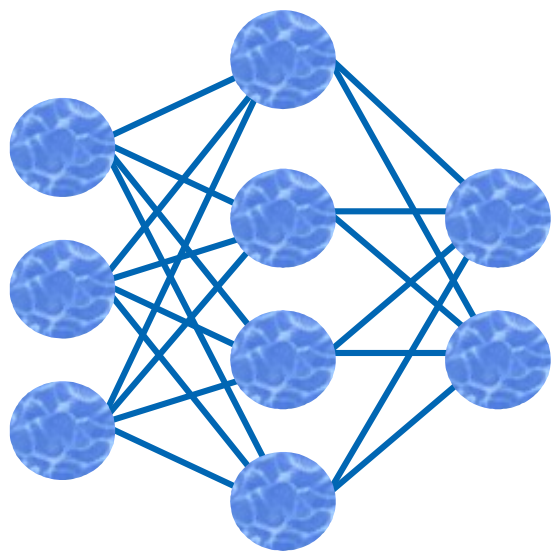
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



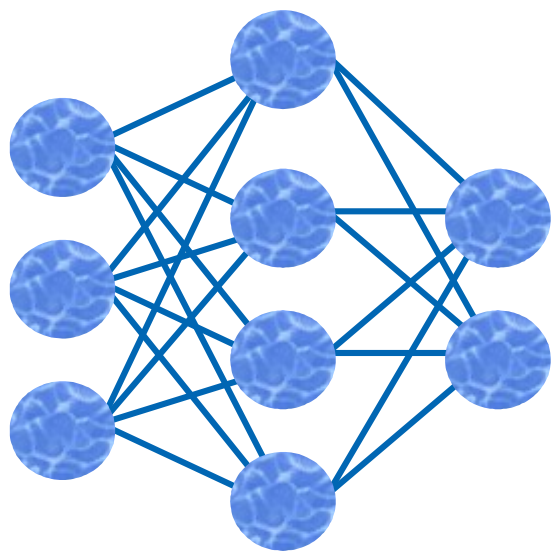
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



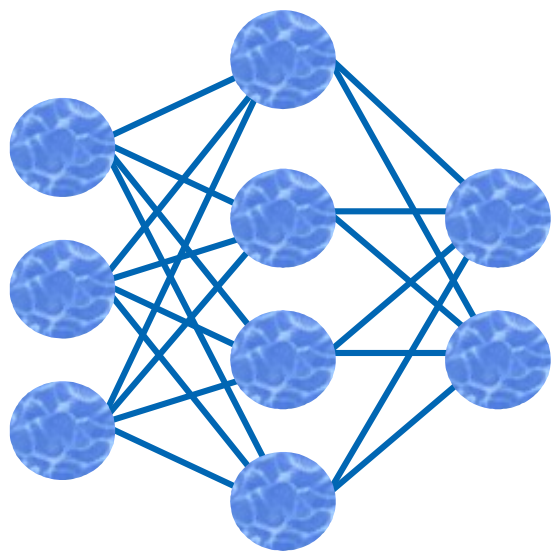
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



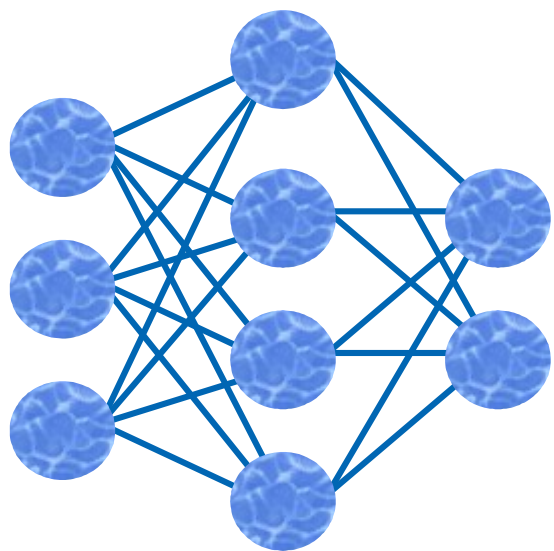
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



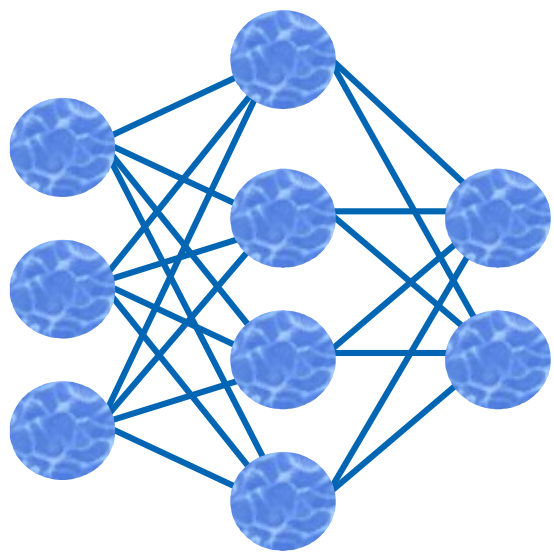
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

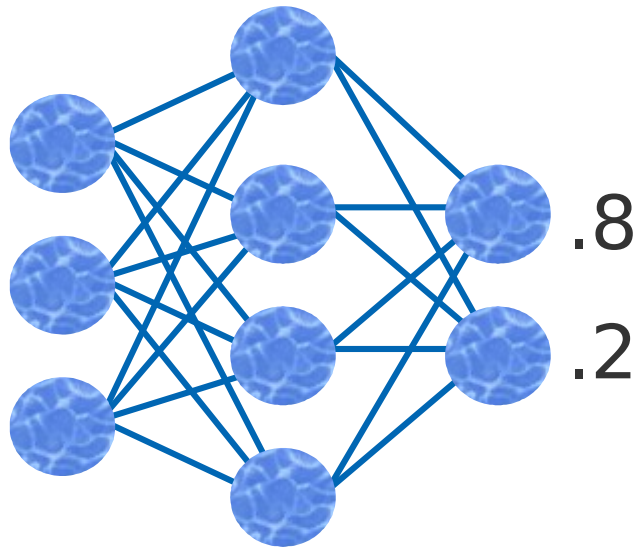


ID	P1	P0
001	0	1
002	1	0
003	0	1



- Get another output

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

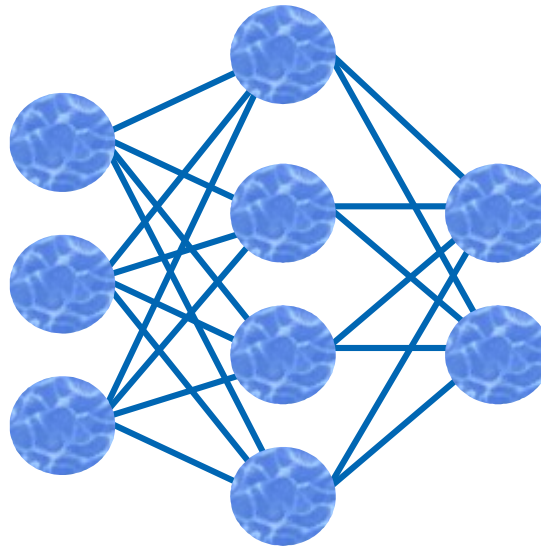


ID	P1	P0
001	0	1
002	1	0
003	0	1



Calculate the error compared to the corresponding labels

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



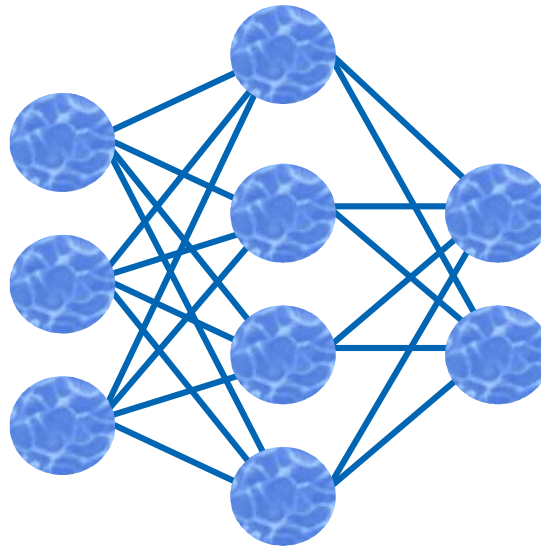
E
r
r
o
r

ID	P1	P0
001	0	1
002	1	0
003	0	1



Go backwards and carefully change the weights in the network again

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

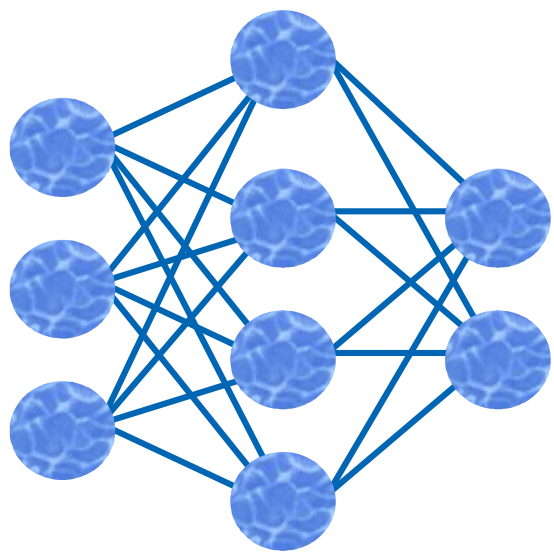


E
r
r
o
r

ID	P1	P0
001	0	1
002	1	0
003	0	1



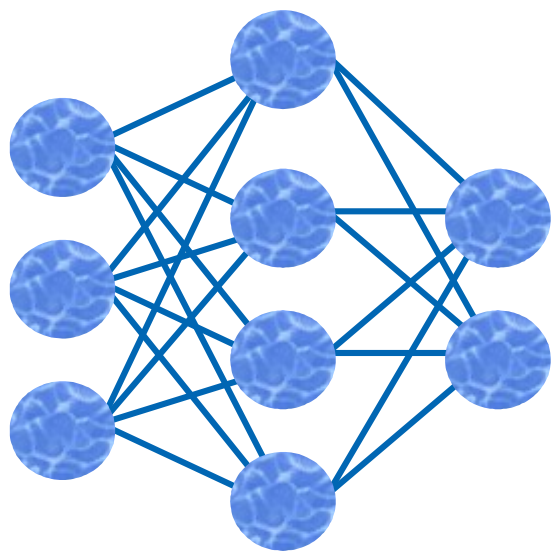
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



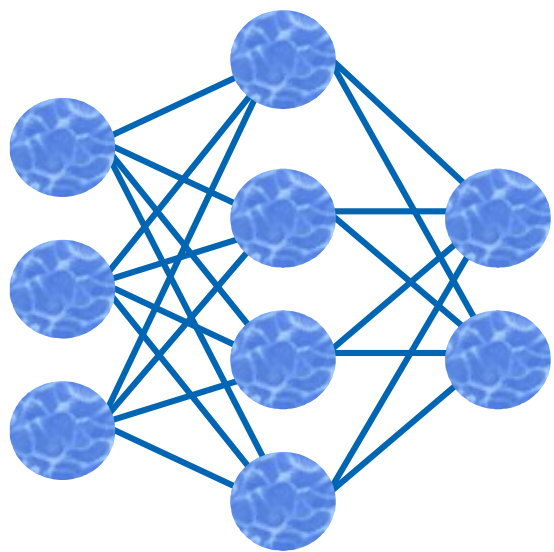
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



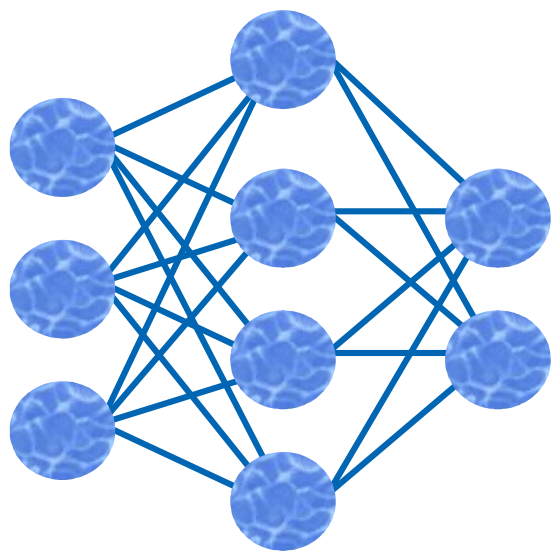
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



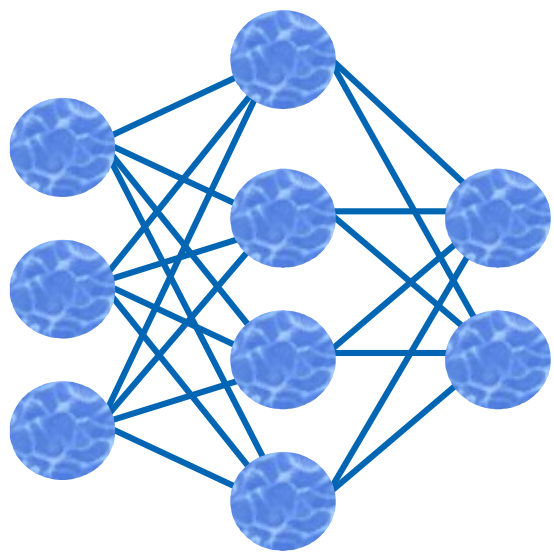
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1



ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

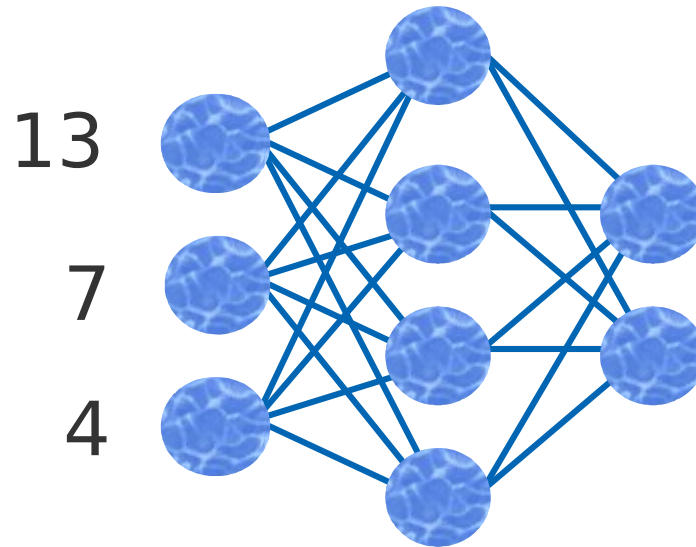


ID	P1	P0
001	0	1
002	1	0
003	0	1



After going completely backwards,

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

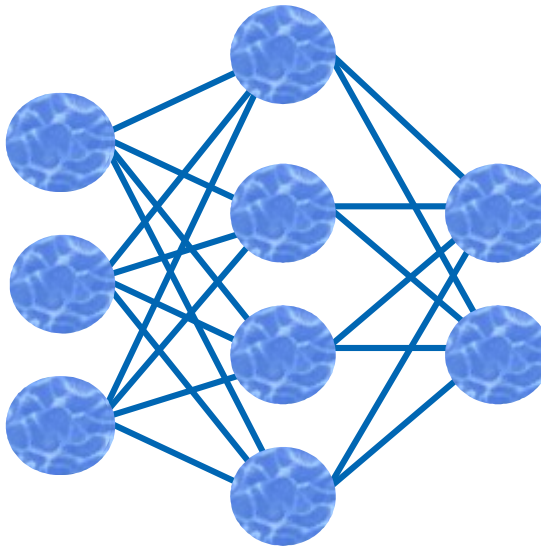


ID	P1	P0
001	0	1
002	1	0
003	0	1



After going completely backwards we go forwards again ... etc... etc

ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...

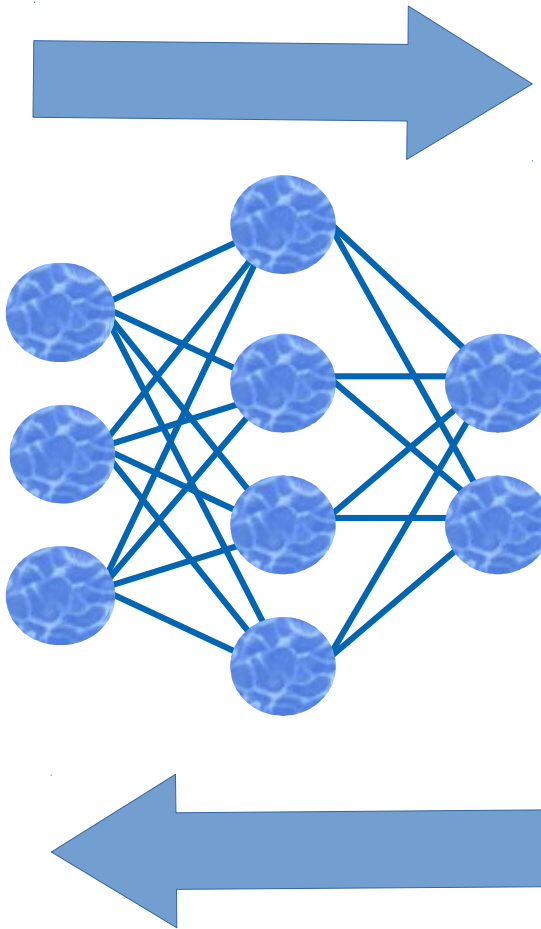


ID	P1	P0
001	0	1
002	1	0
003	0	1



We have to go forwards and backwards many times to make the network smart.

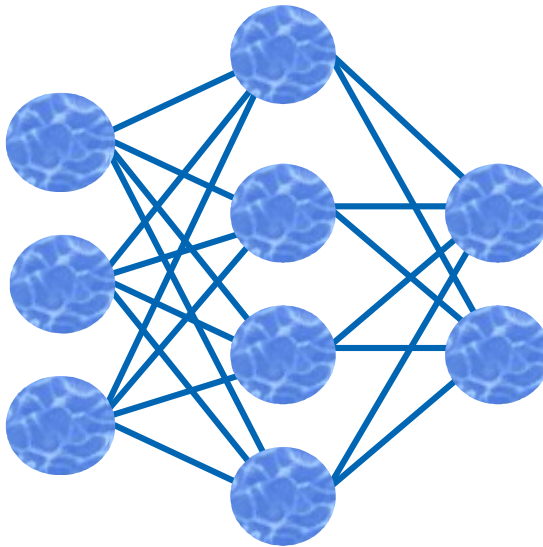
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1

After a while, we stop.

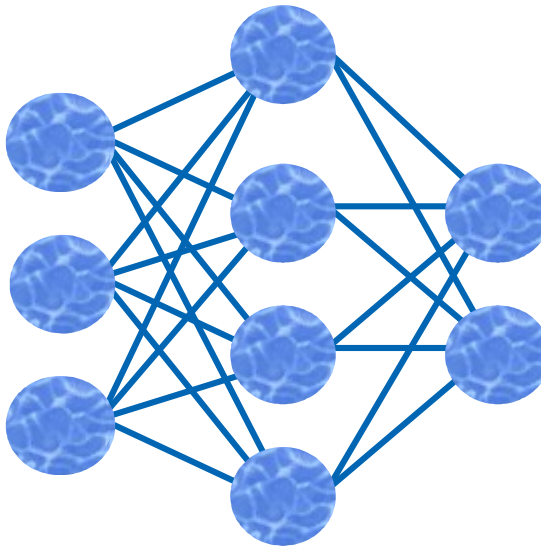
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1

We unhook the network from the training data.

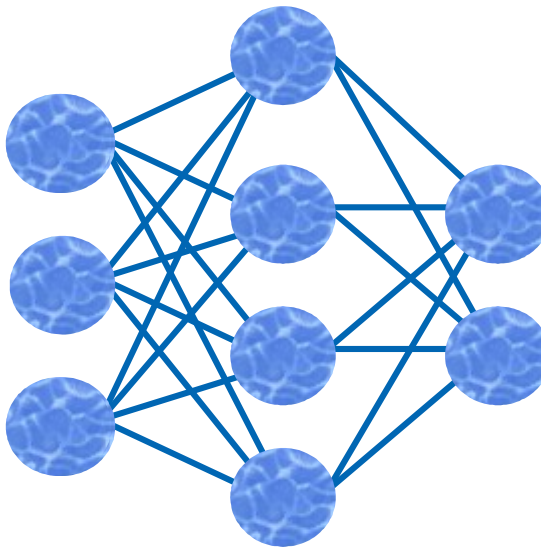
ID	V1	V2	V3
001	1	5	2
002	13	7	4
003	3	0	5
...



ID	P1	P0
001	0	1
002	1	0
003	0	1

We and try it on new data it has never seen before to test it.

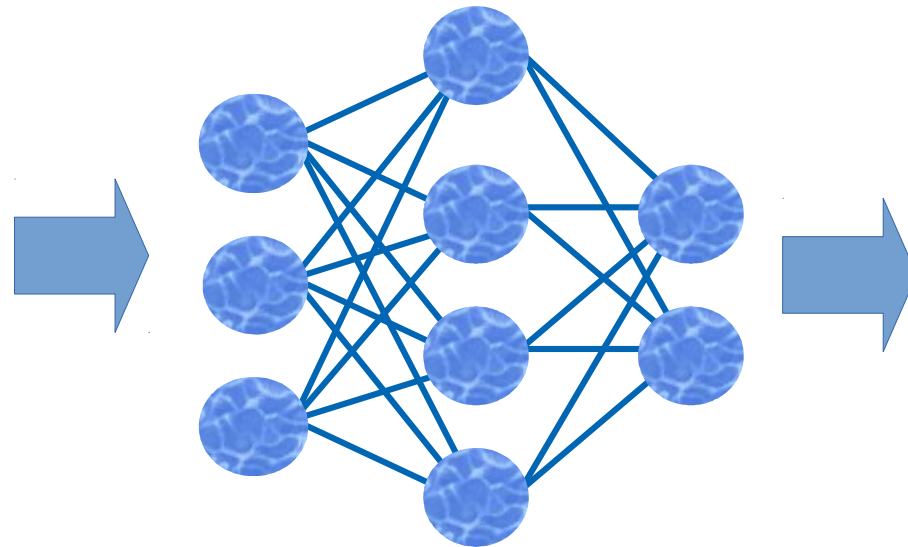
ID	V1	V2	V3
001	3	8	5
002	11	11	2
003	7	2	0
...



ID	P1	P0
001	1	0
002	1	0
003	0	1

Without any training, we feed all the new data through the network forward to see if it is close to the labeled data

ID	V1	V2	V3
001	3	8	5
002	11	11	2
003	7	2	0
...



ID	P1	P0
001	1	0
002	1	0
003	0	1

We take the predictions and the labels, and see how the network did.

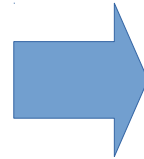
Label	Predicted
1	0.8
0	.4
1	0.9
...	...



Accuracy 93%

The idea here is this is hopefully about how well the network will do when you actually use it for real-life predictions

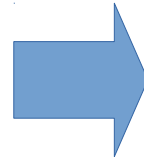
Label	Predicted
1	0.8
0	.4
1	0.9
...	...



Accuracy 93%

So, if we use this neural network to actually award loans, we will hope to know if we are getting paid back or not with a 93% accuracy.

Label	Predicted
1	0.8
0	.4
1	0.9
...	...

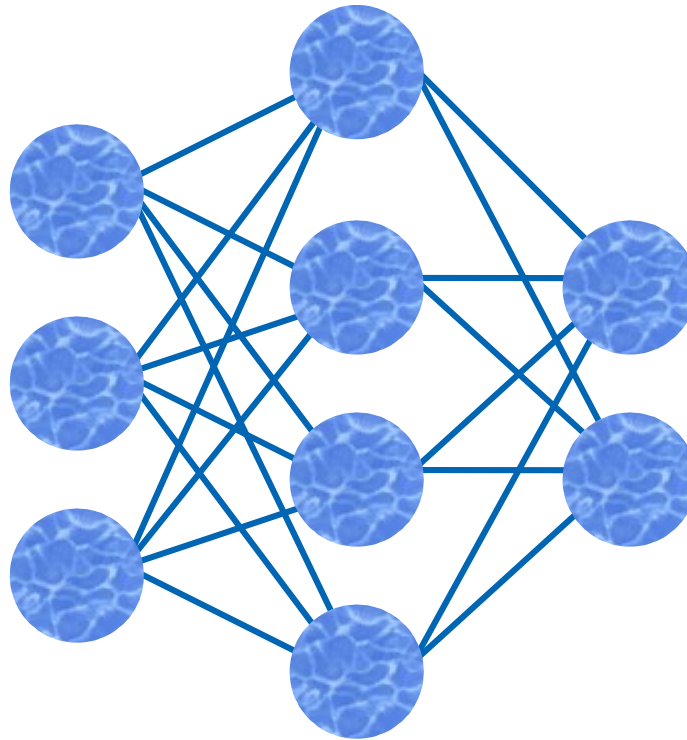


Accuracy 93%

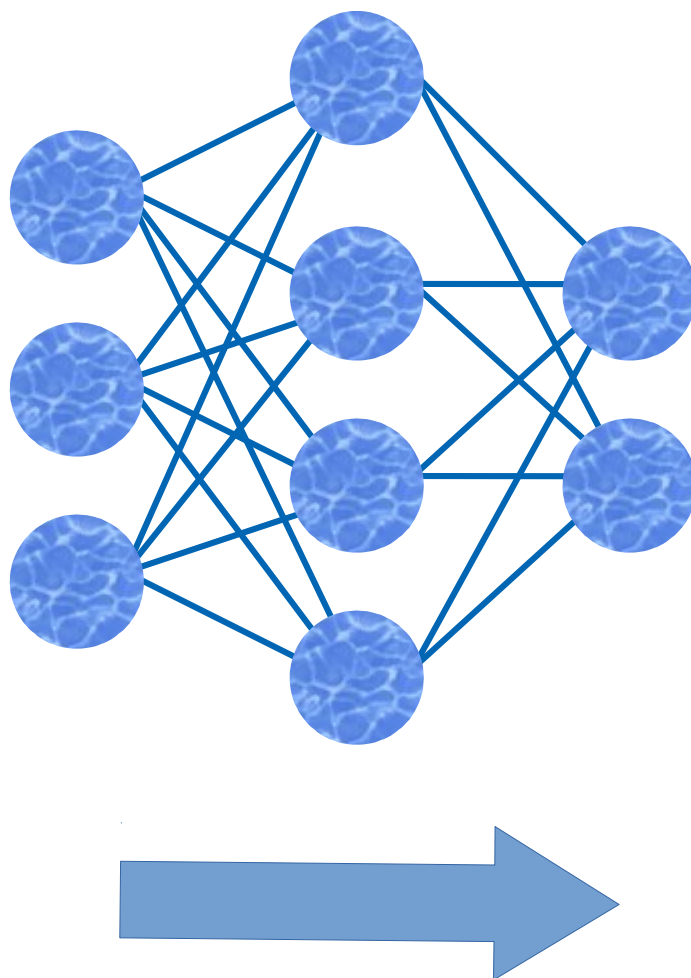
OK, so thats the basics.

OK, so thats the basics. Need a break? Any Questions?

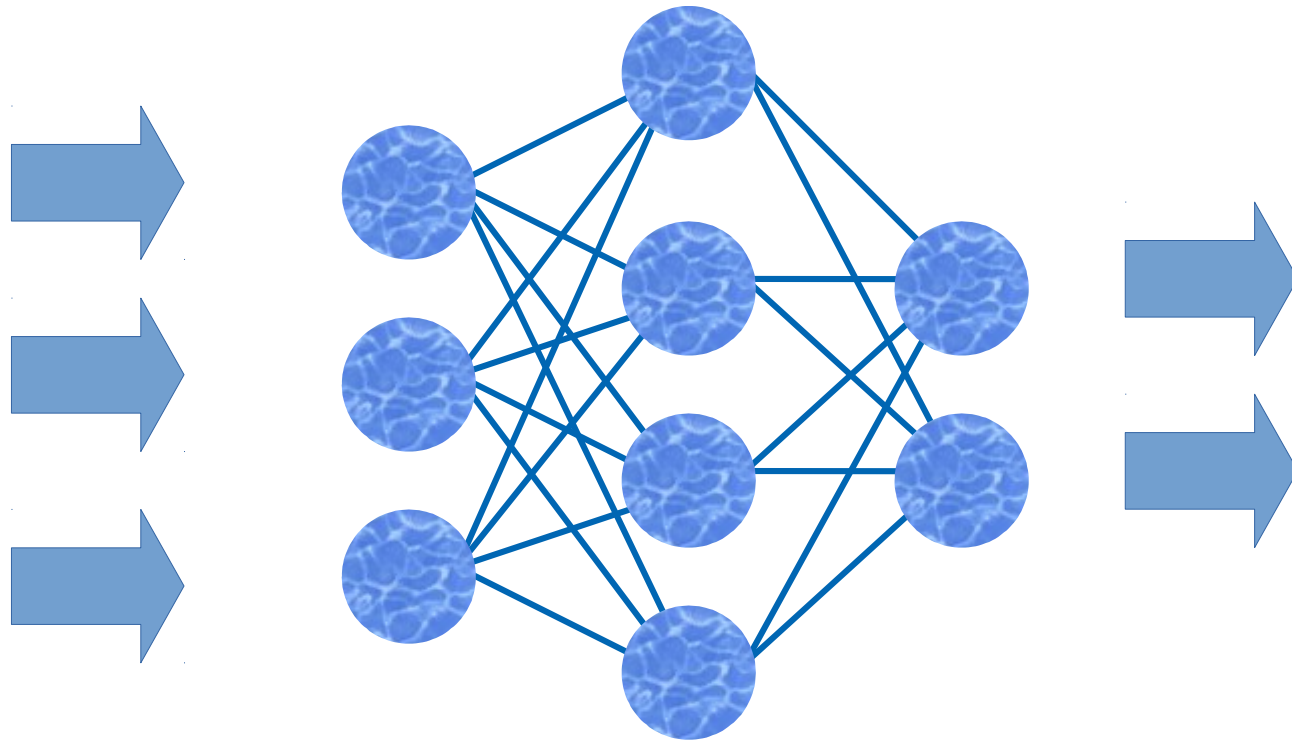
But what actually happens when we go back and forth over the network?



We will dive in and find out what actually happens in the forward part, its called a **forward pass**.

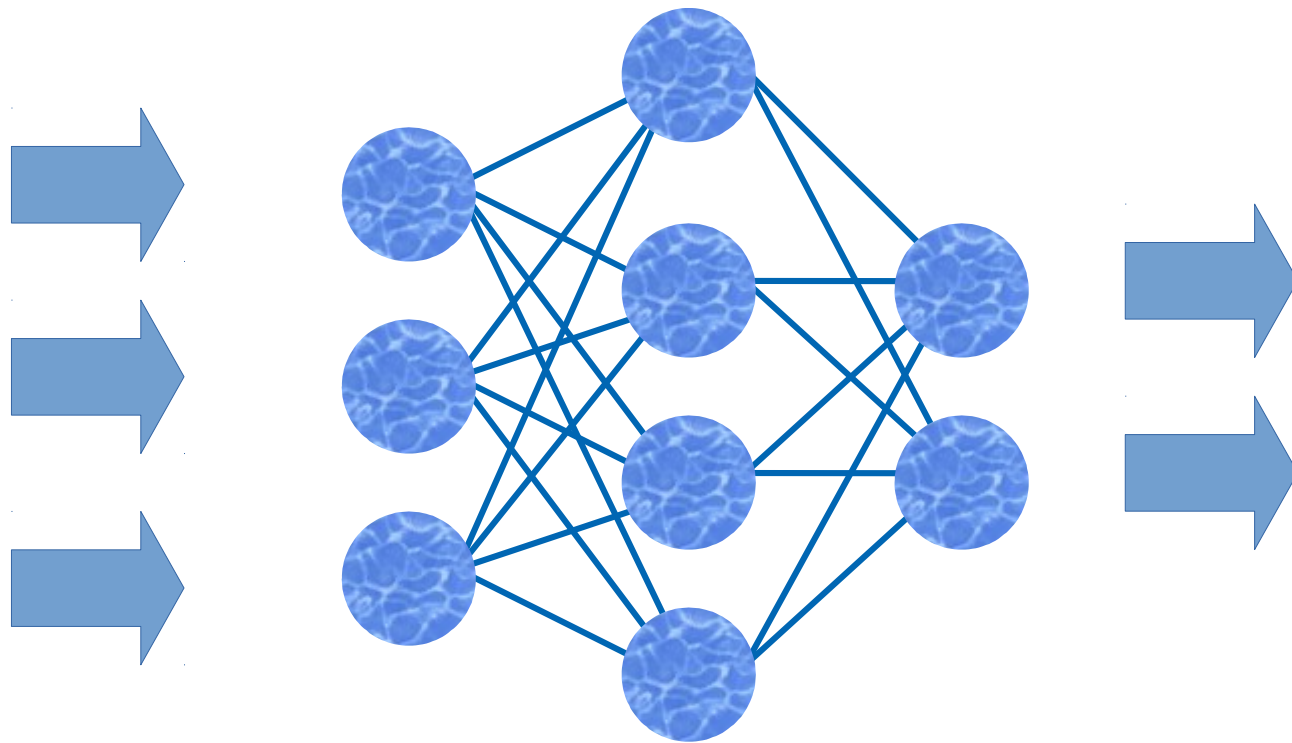


This example neural network has 3 inputs dots (where inputs go) and 2 output dots (where outputs go)



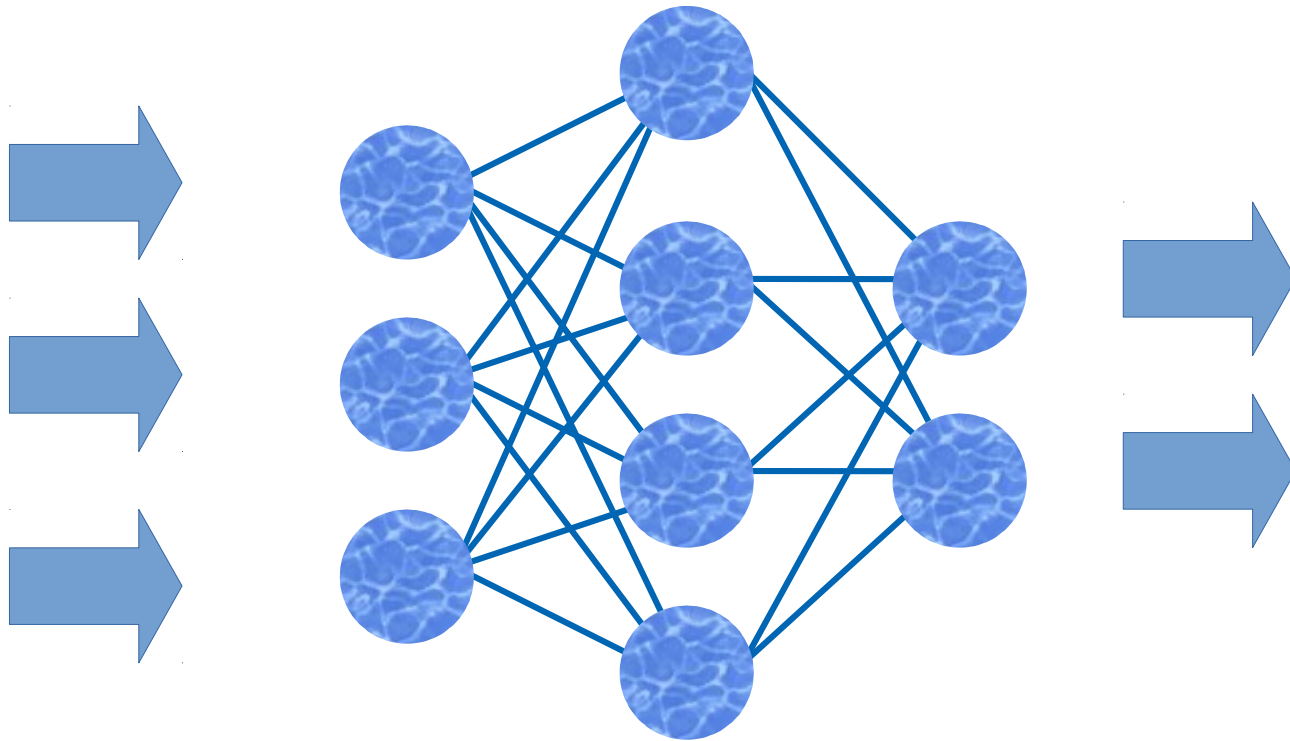
Also, there are 4 middle dots.

What's going in and going out?



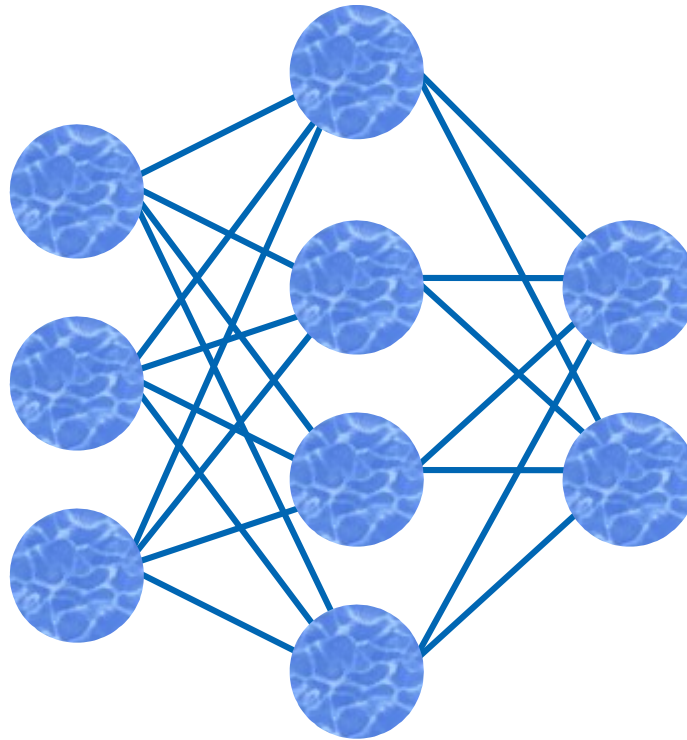
Numbers. 3 numbers in, 2 numbers out for a single forward pass.

Wait, what about the middle dots?



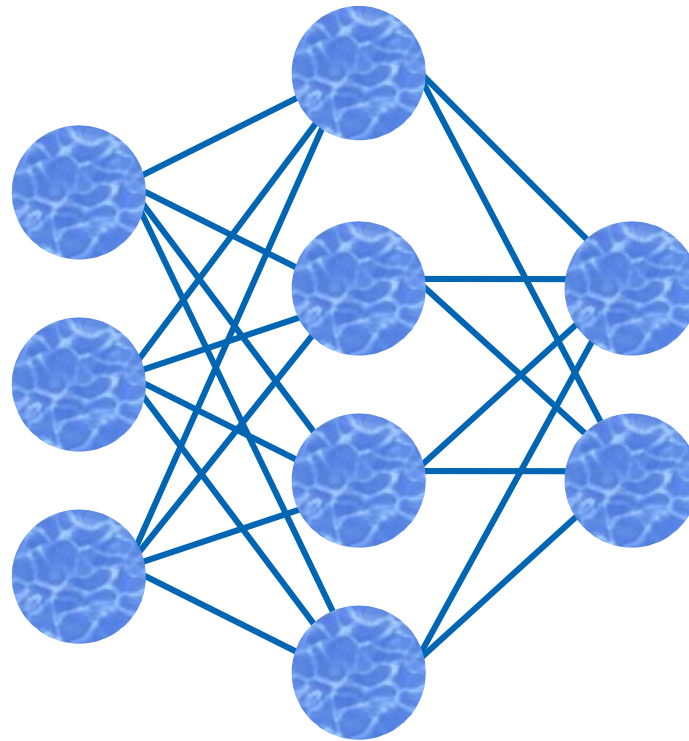
The 3 numbers become 4 numbers, and then 2 numbers

Lots of dot talk so far, what about the lines?



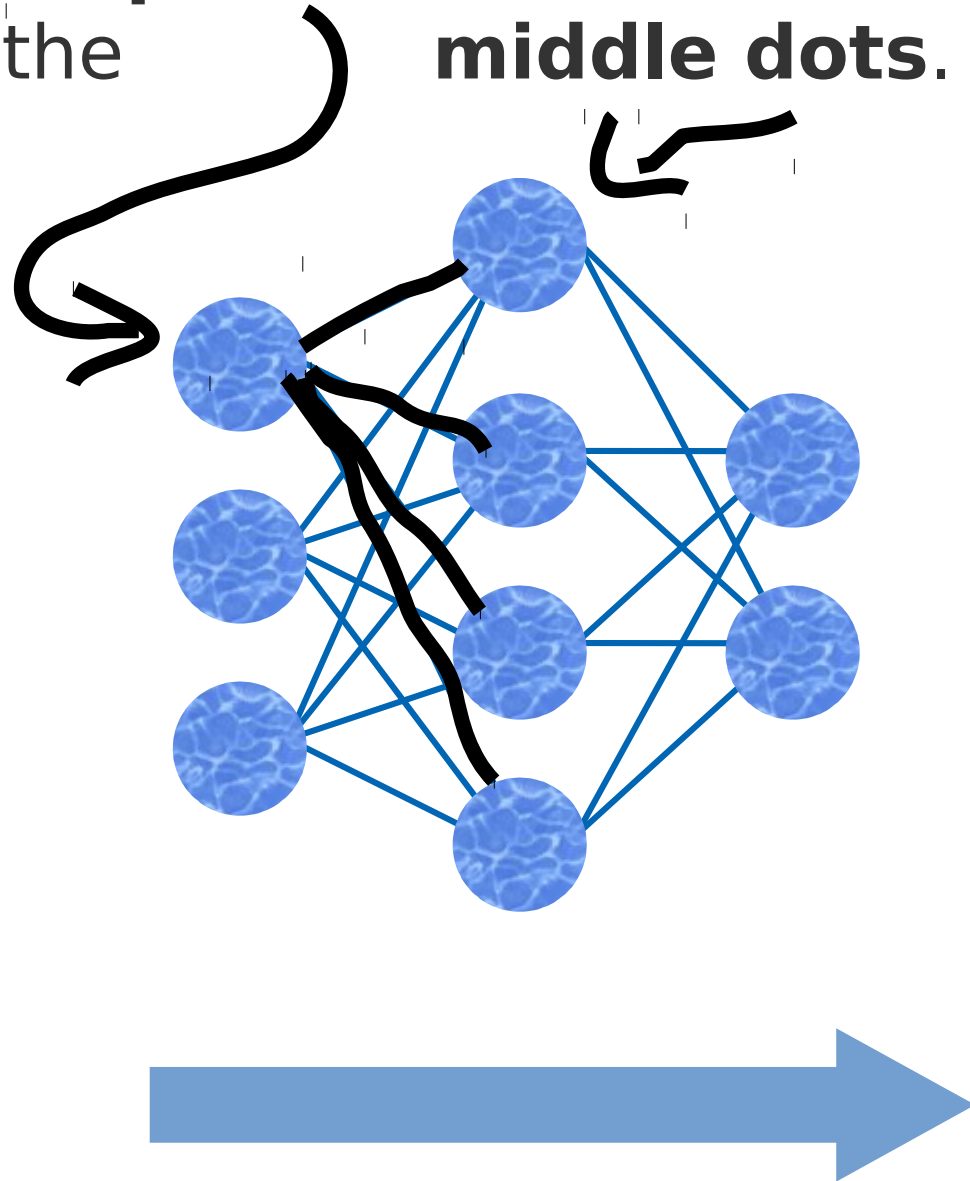
The lines represent the flow of numbers through the network

Think of the lines as little arrows with the pointy side on the right



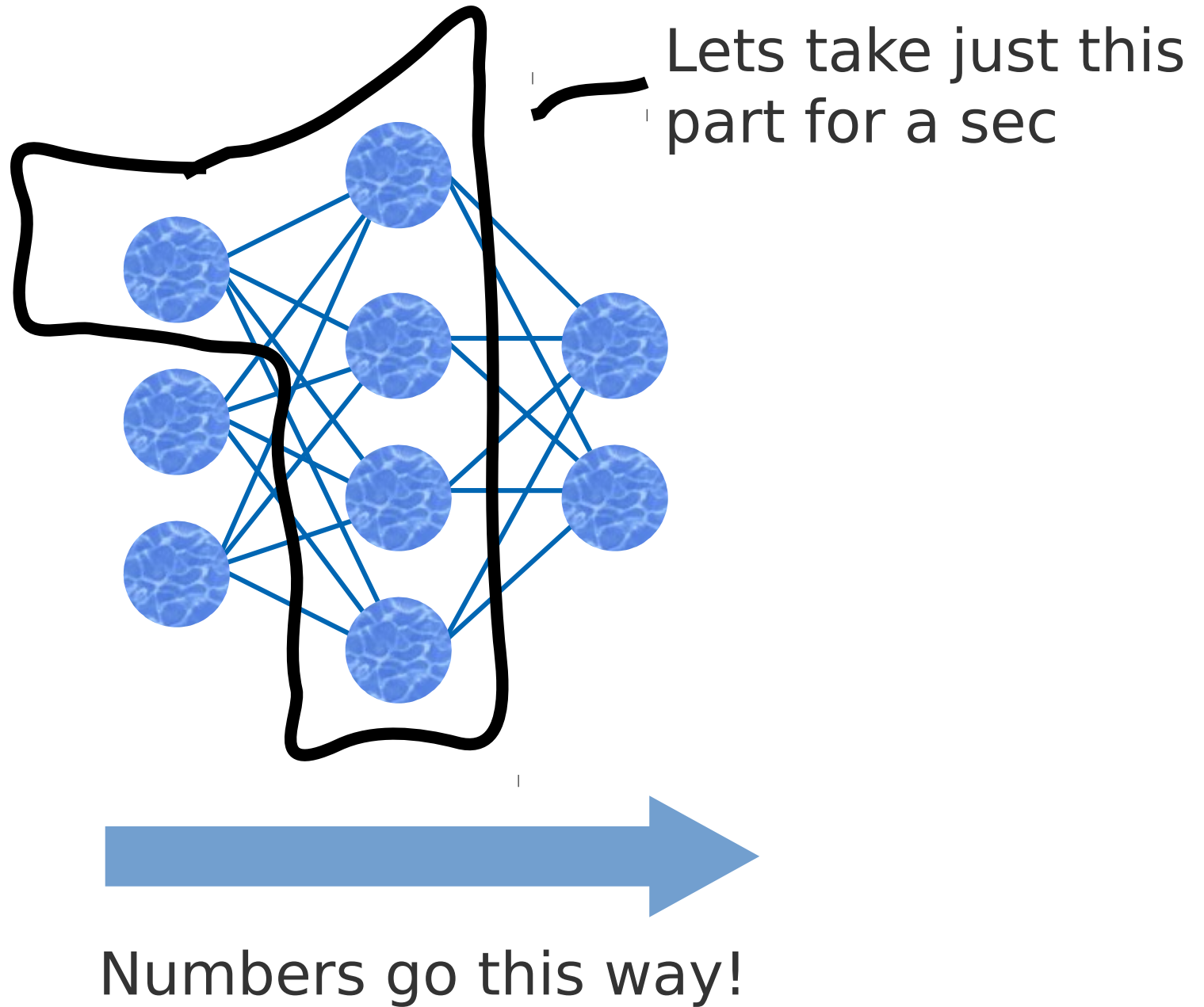
Numbers go this way!

So, each **input number** “flows” down the lines to each of the **middle dots**.

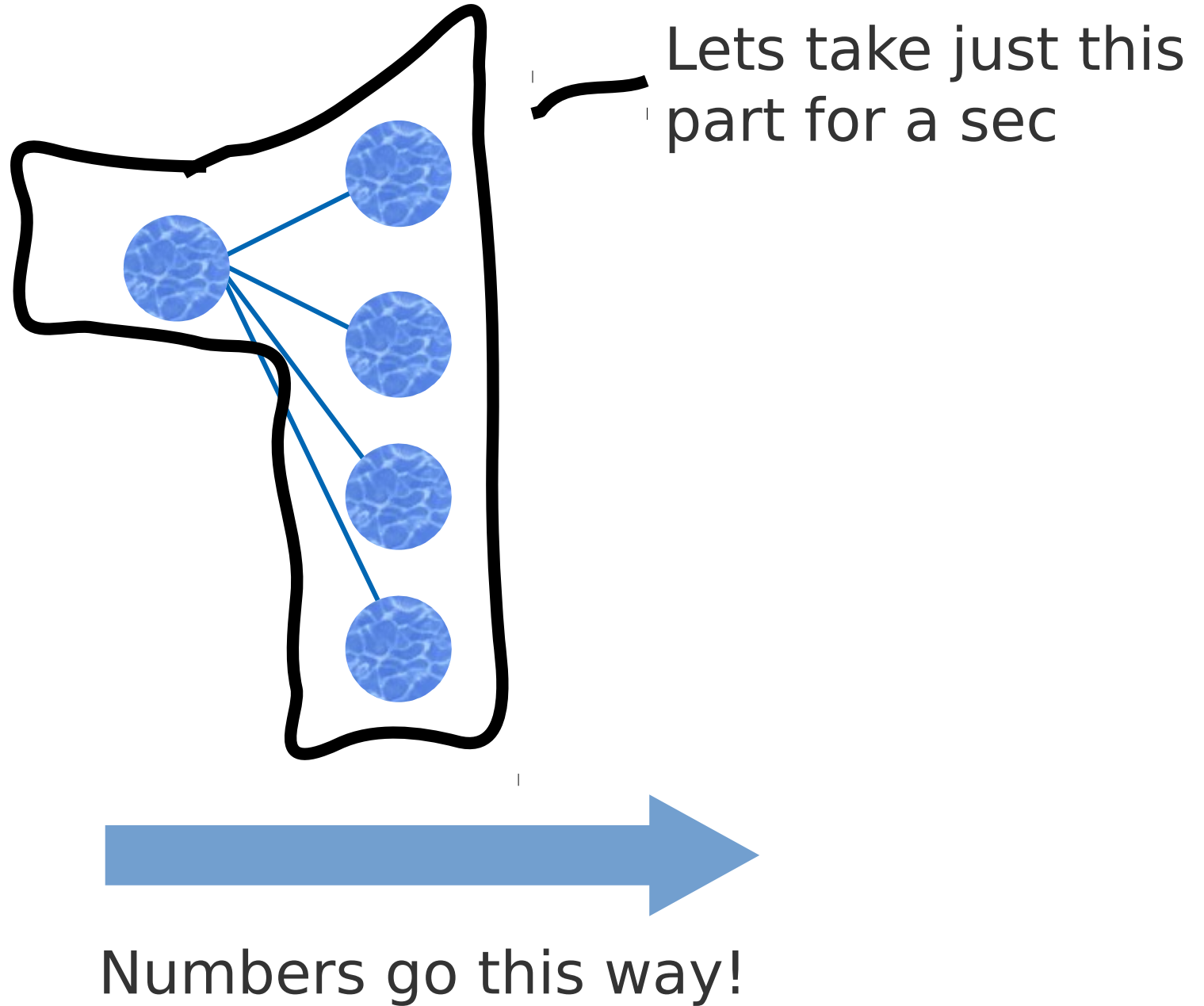


Numbers go this way!

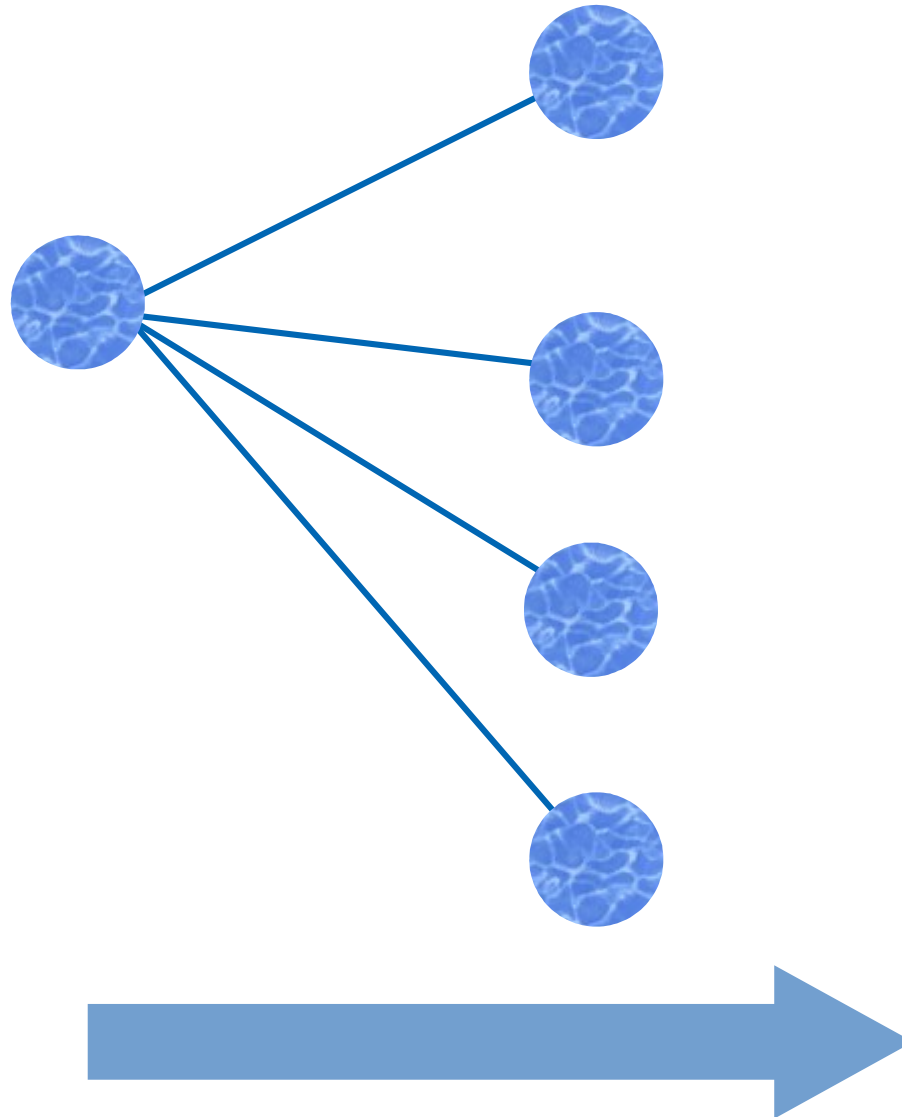
But something else happens along the way.



But something else happens along the way.

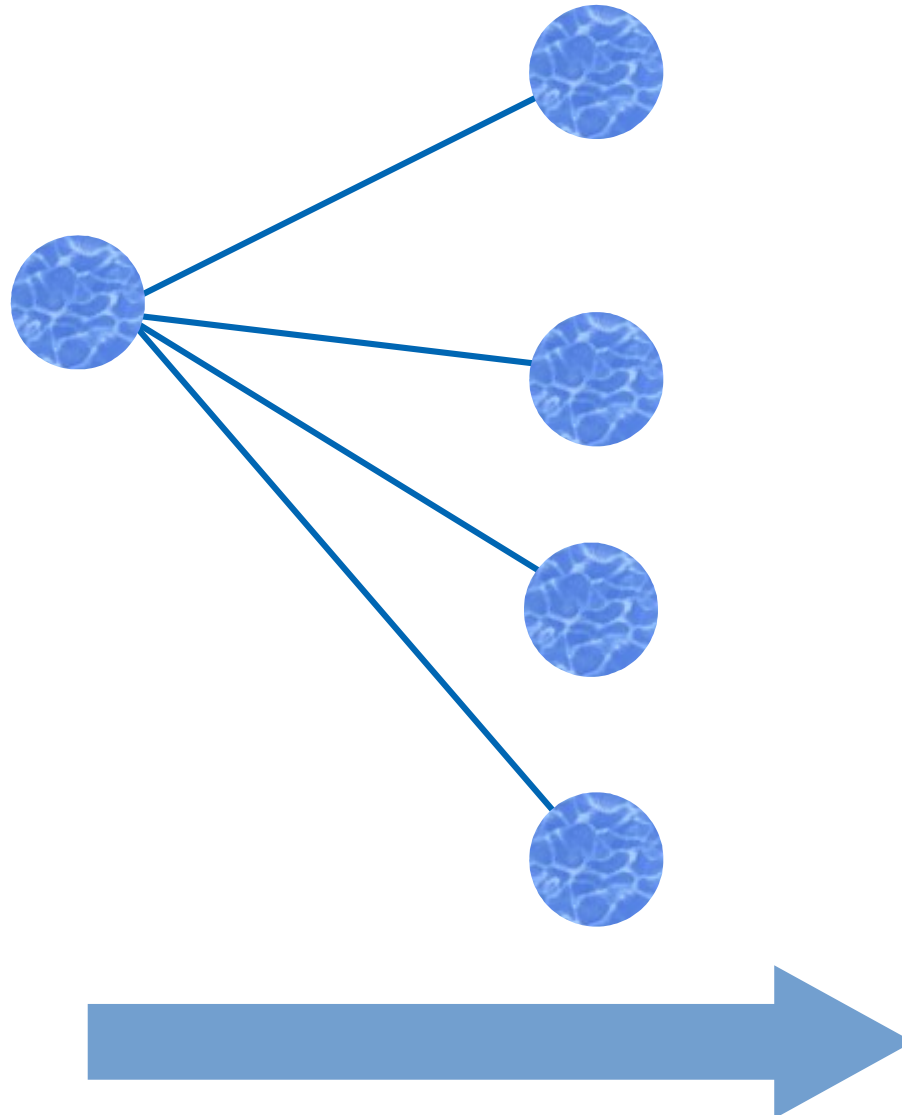


And spread out a bit ...



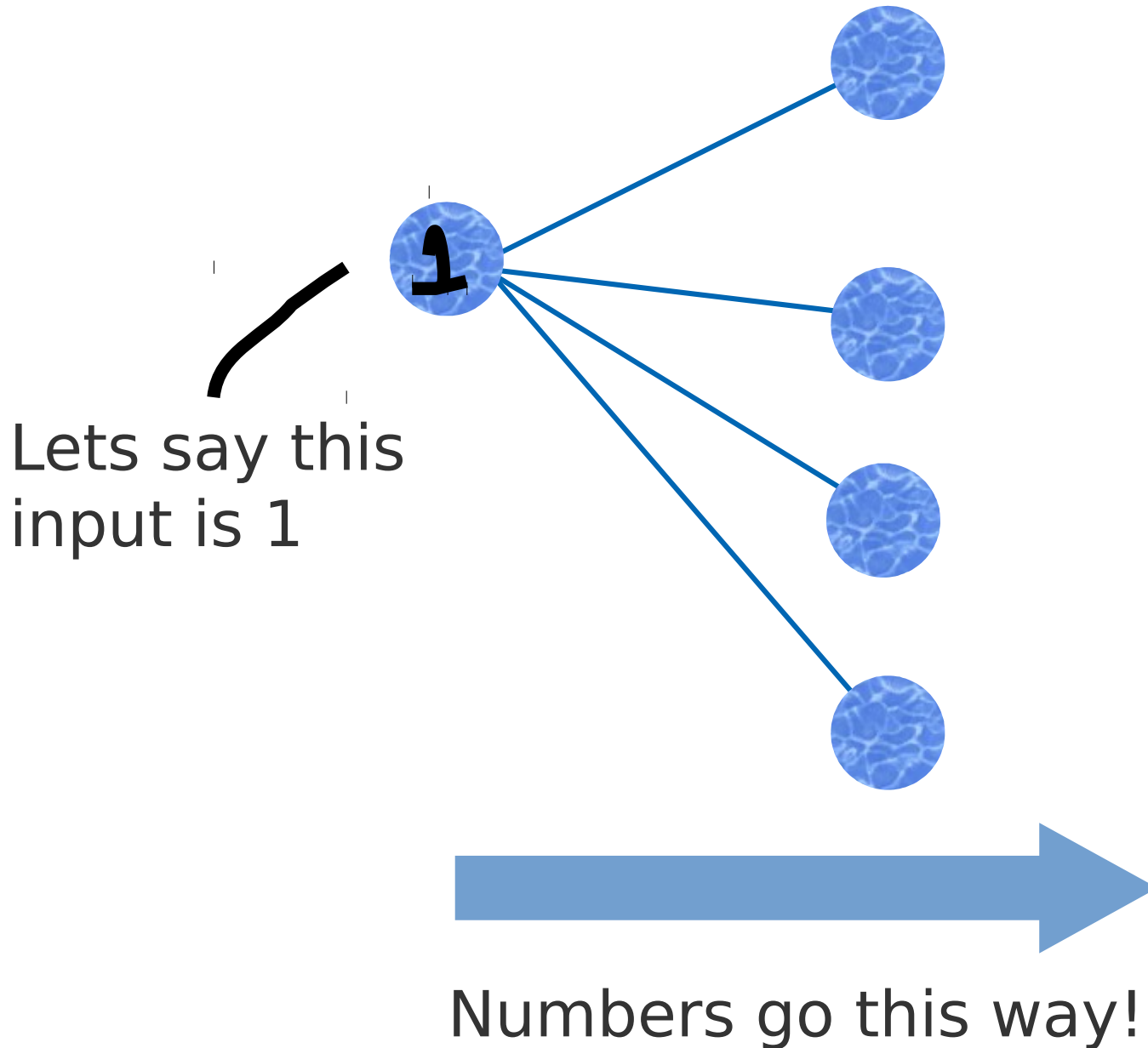
Numbers go this way!

So what else is happening? The input number is multiplied by a new number along the way.

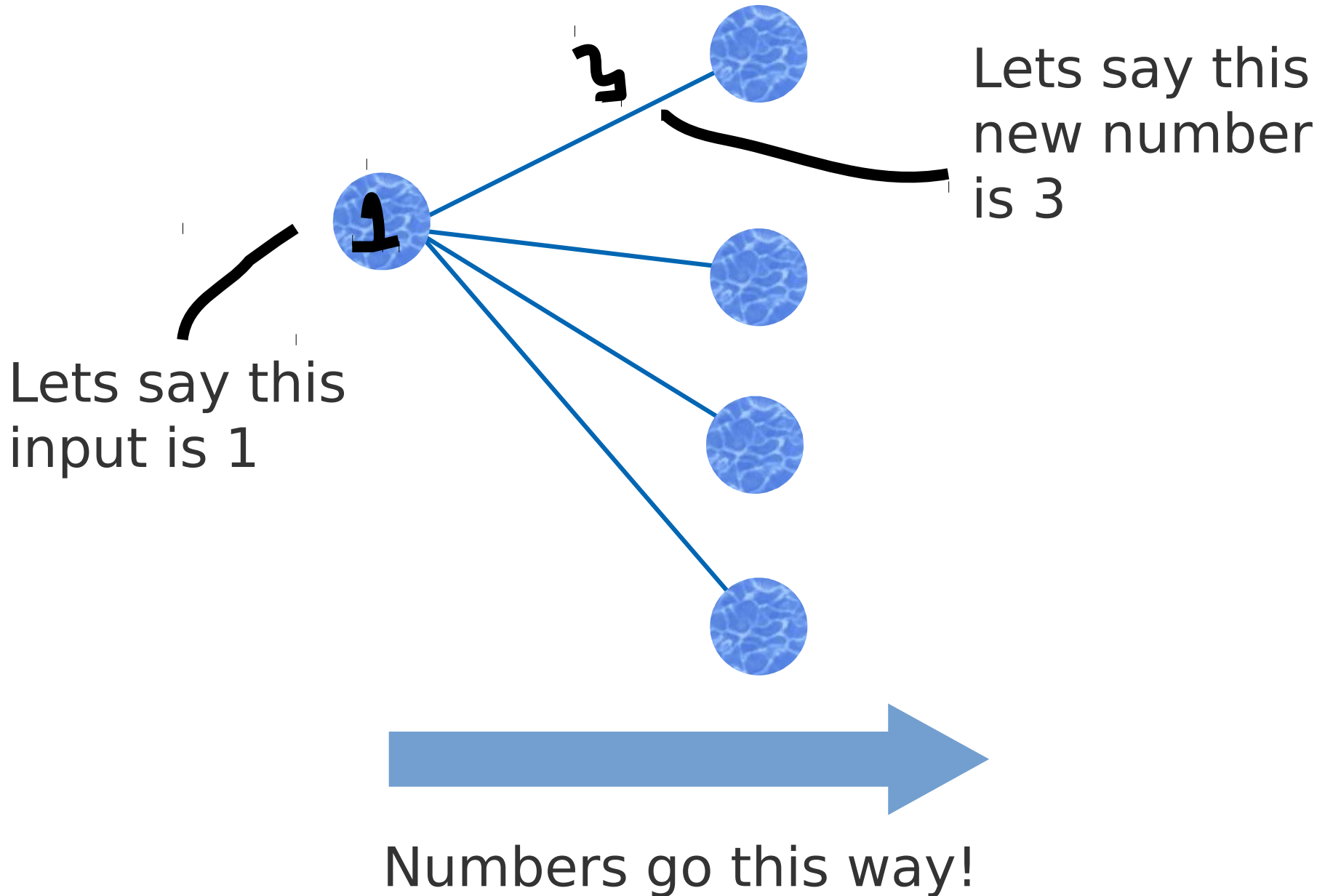


Numbers go this way!

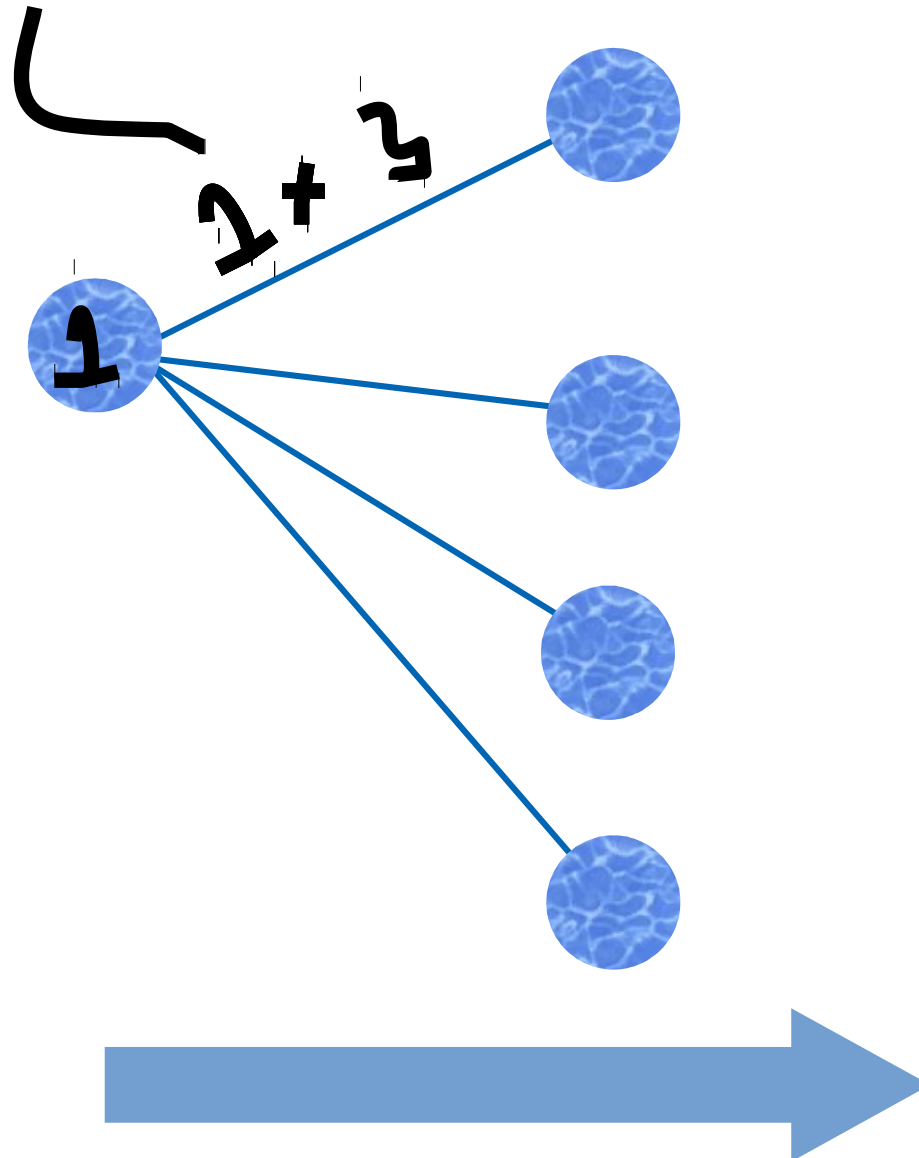
So what else is happening? Each number is multiplied by a new number along the way.



So what else is happening? Each number is multiplied by a new number along the way.

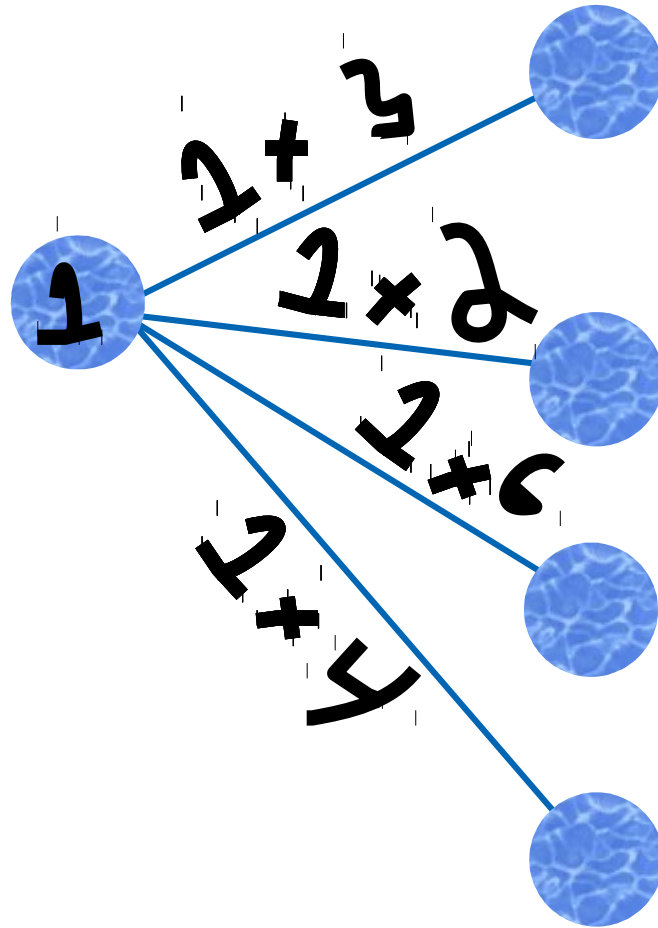


Those numbers are multiplied and the result flows forward.



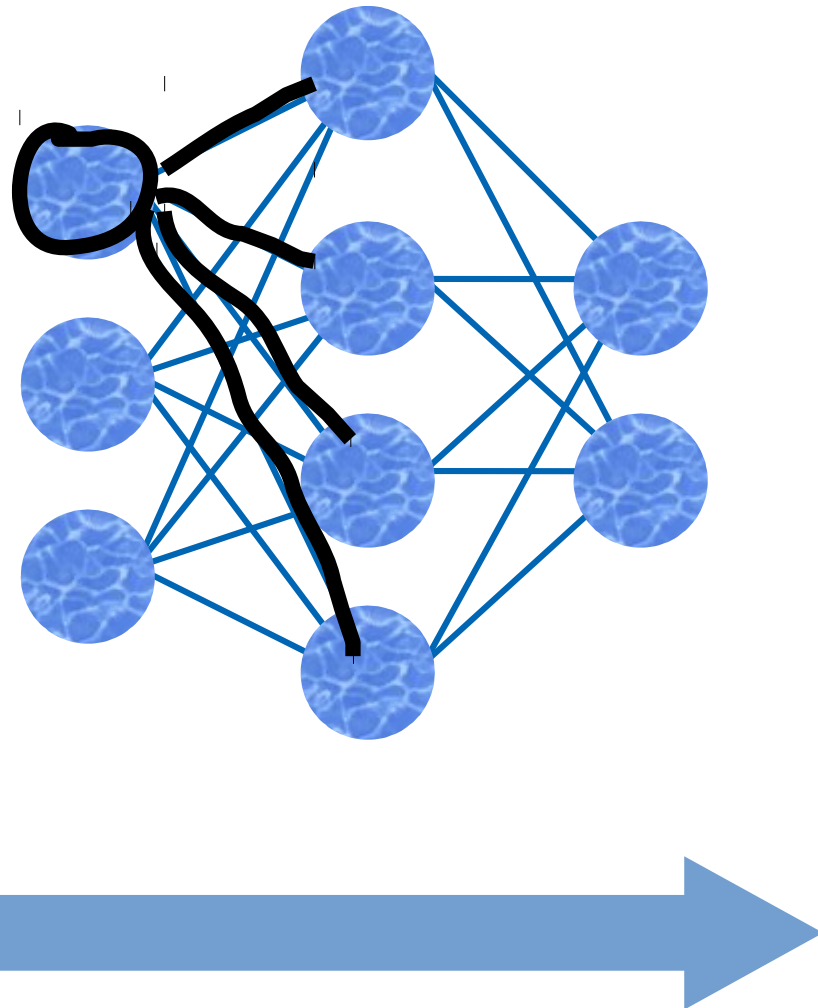
Numbers go this way!

The same thing happens down each line, the input staying the same, but the multiplier is unique for each line



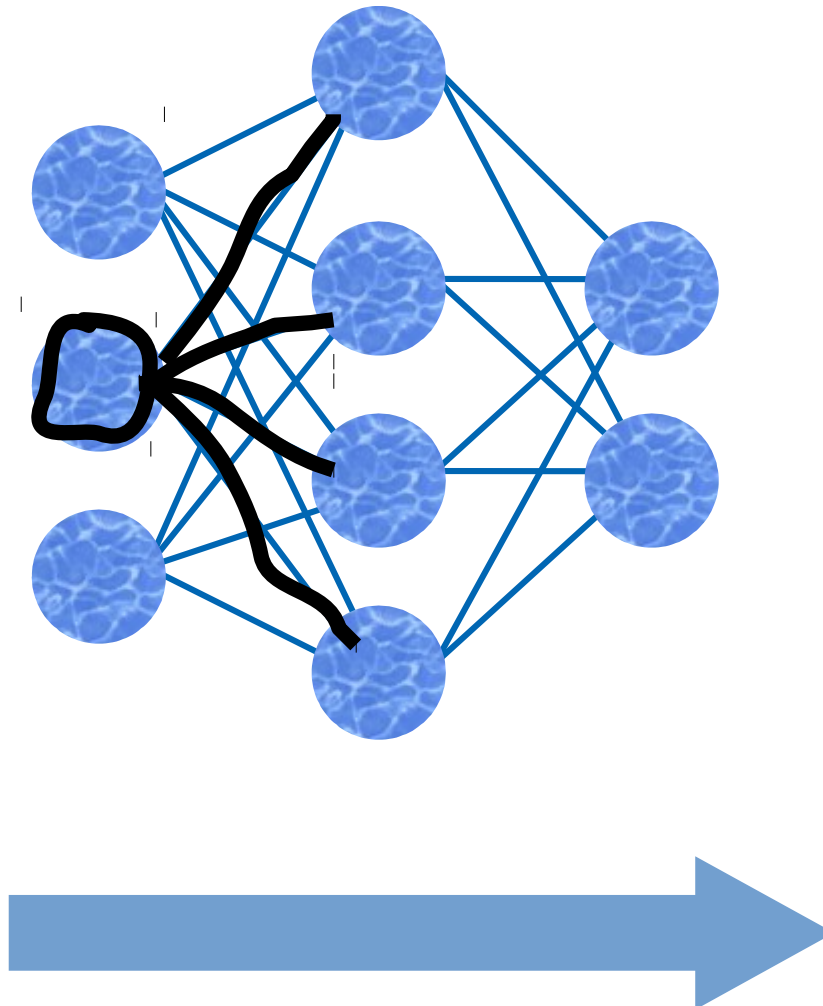
Numbers go this way!

We just looked at what happens in this part



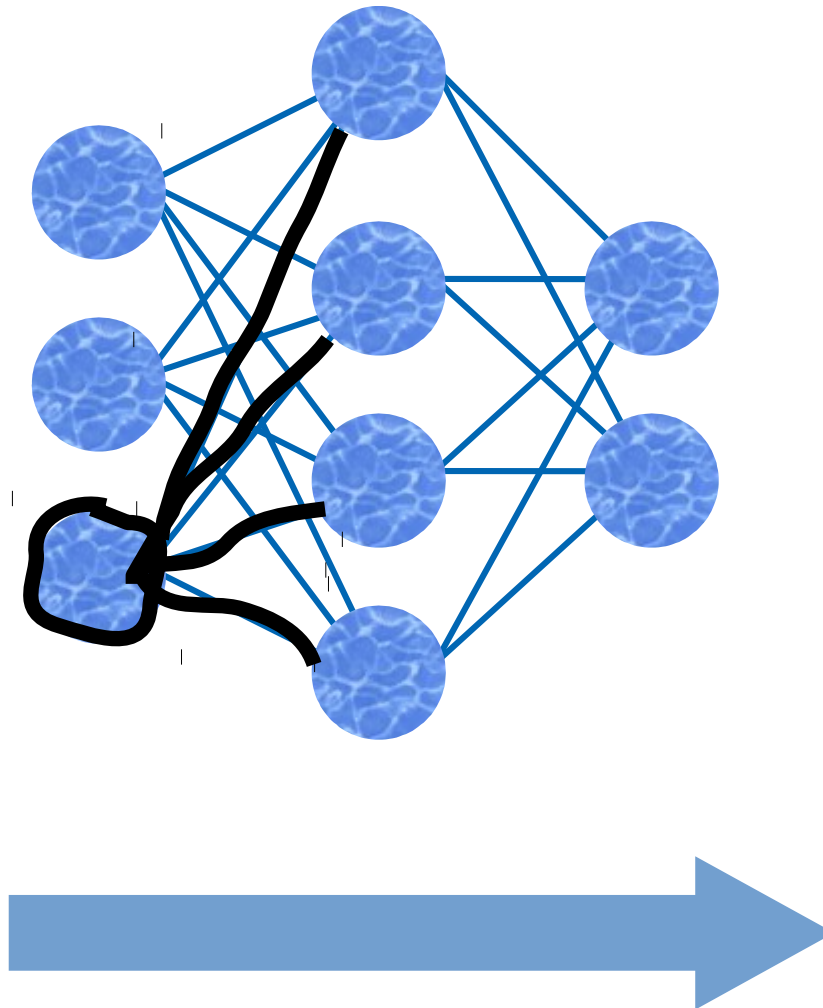
Numbers go this way!

That process is repeated for each input dot with unique multiplier numbers for each line.



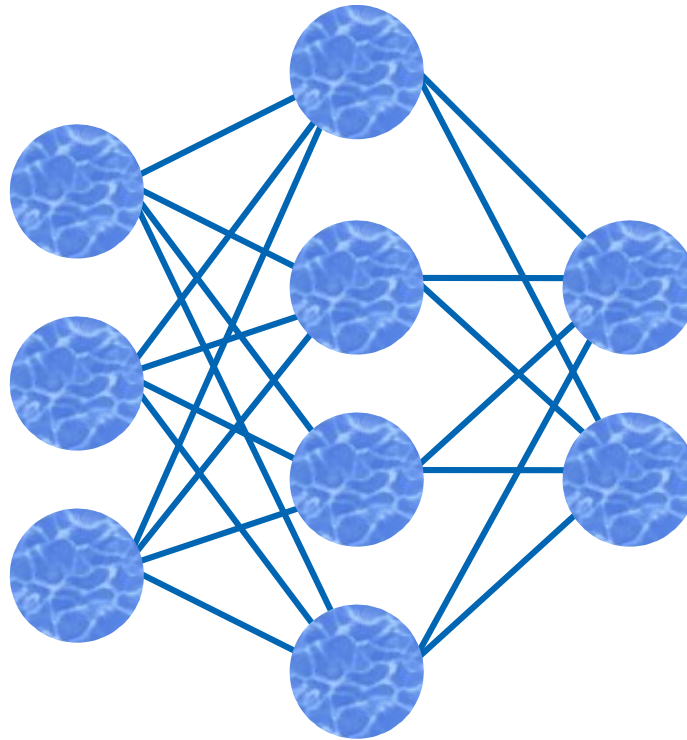
Numbers go this way!

That process is repeated for each input dot with unique multiplier numbers for each line.



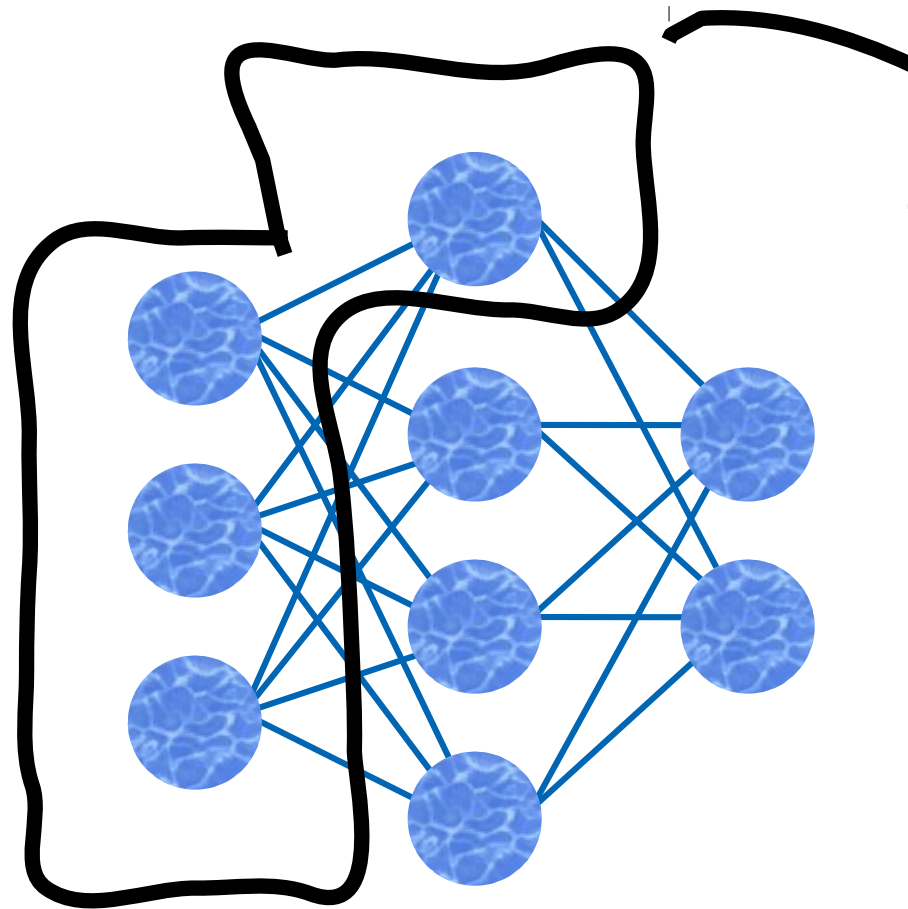
Numbers go this way!

What happens next?



Numbers go this way!

What happens next?

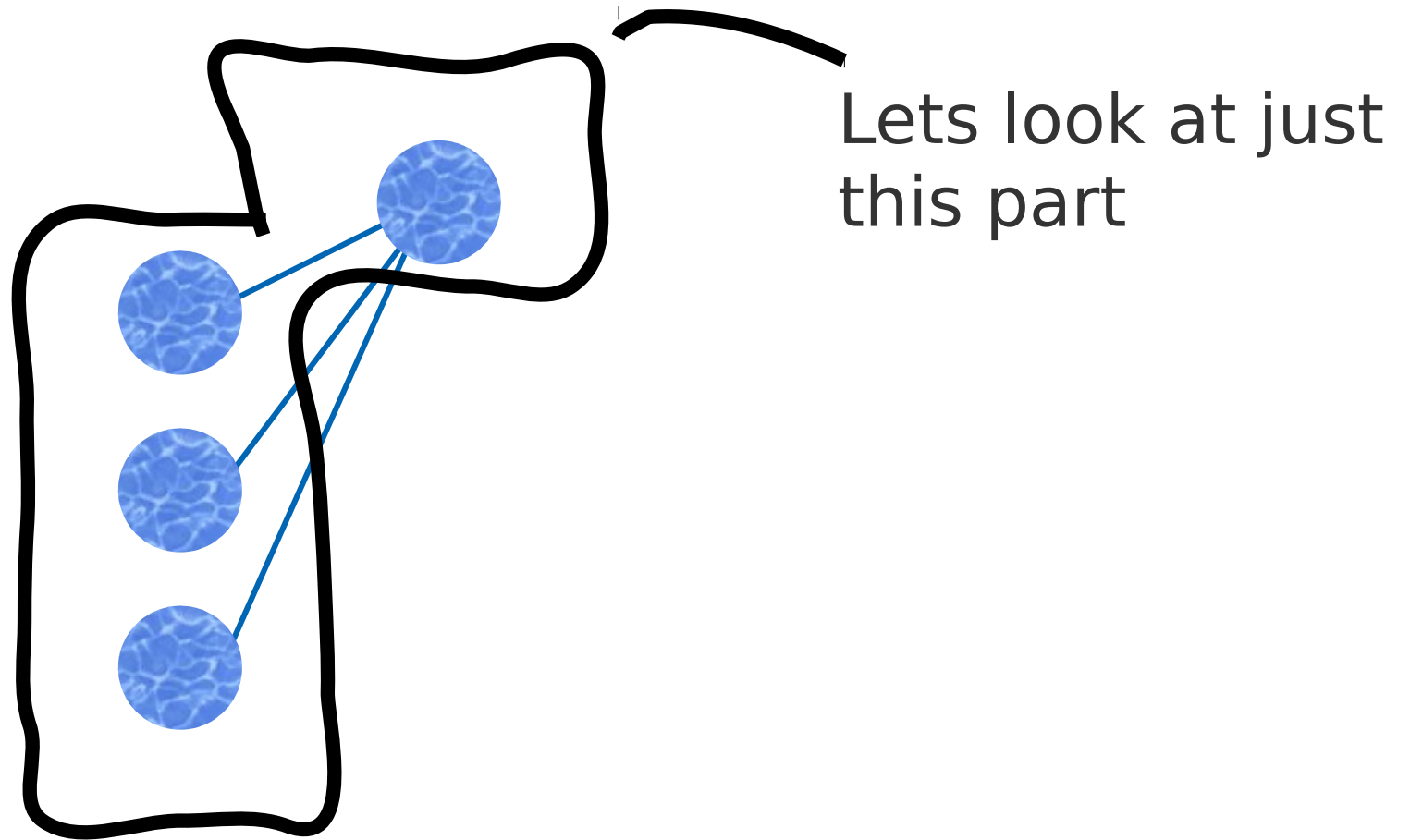


Lets look at just
this part



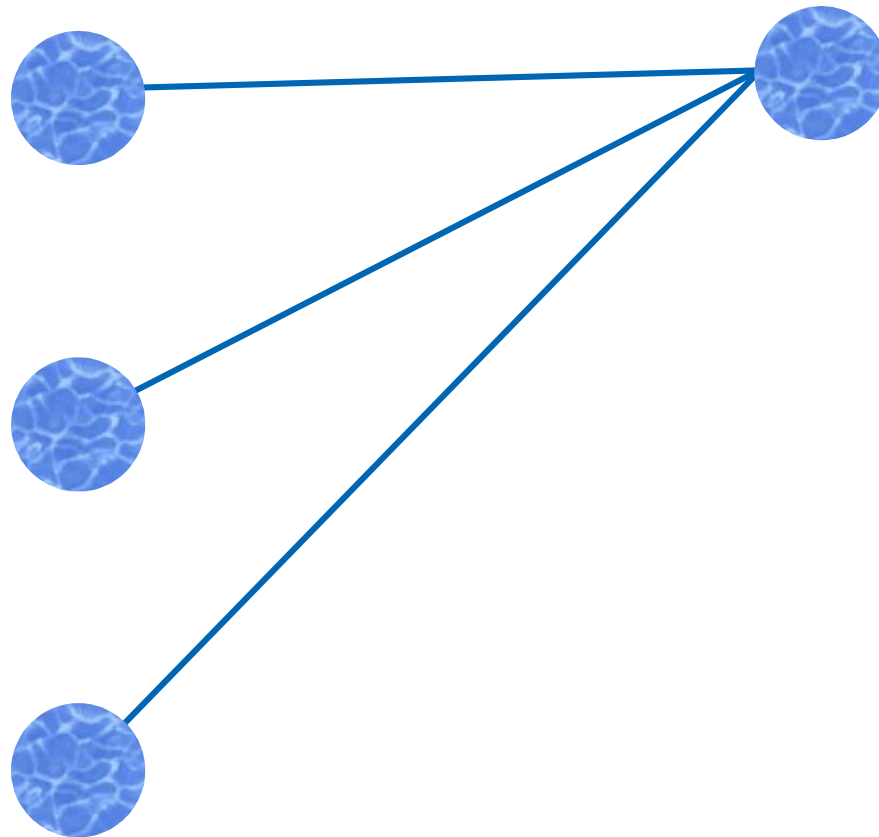
Numbers go this way!

What happens next?



Numbers go this way!

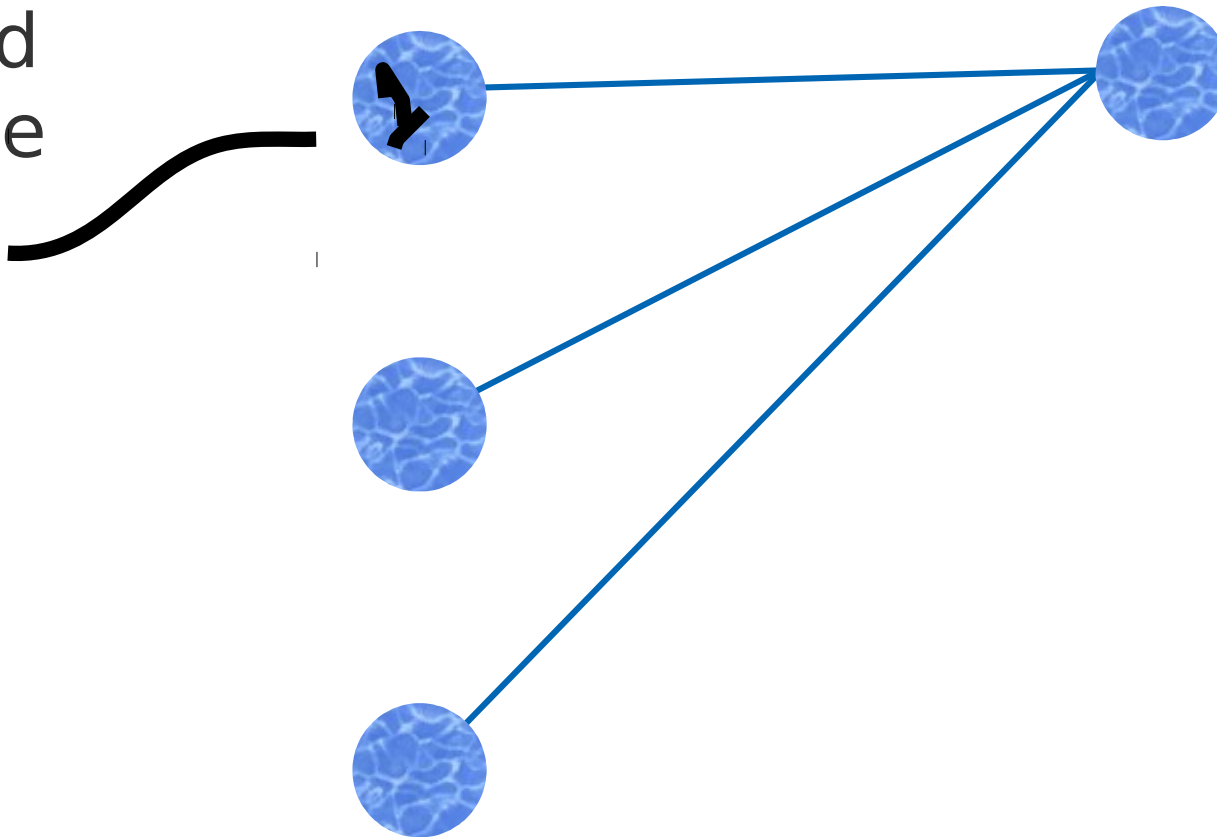
And spread it out...



Numbers go this way!

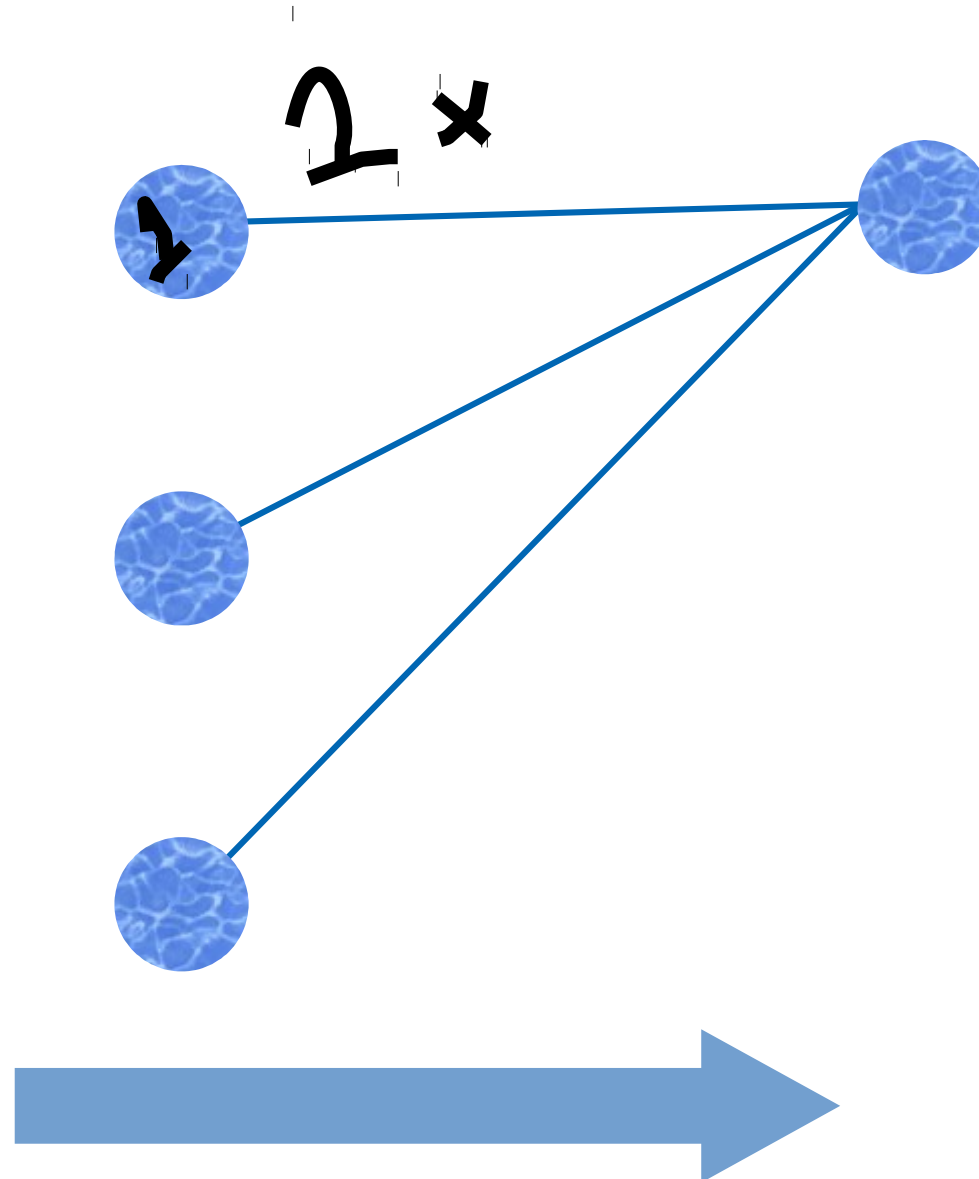
Remember, each line is an input times a multiplier number

We said
this one
was 1



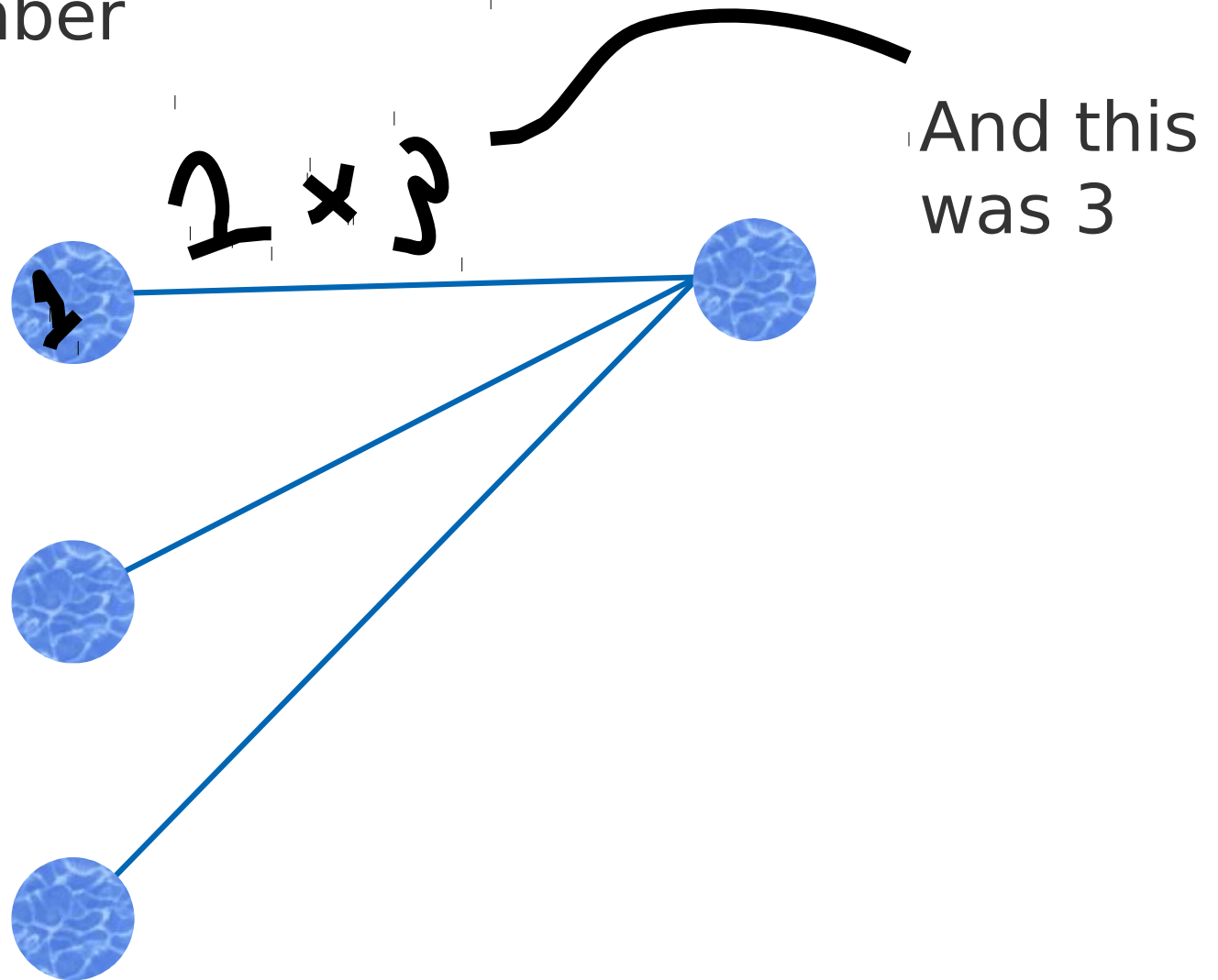
Numbers go this way!

Remember, each line is an input times a multiplier number

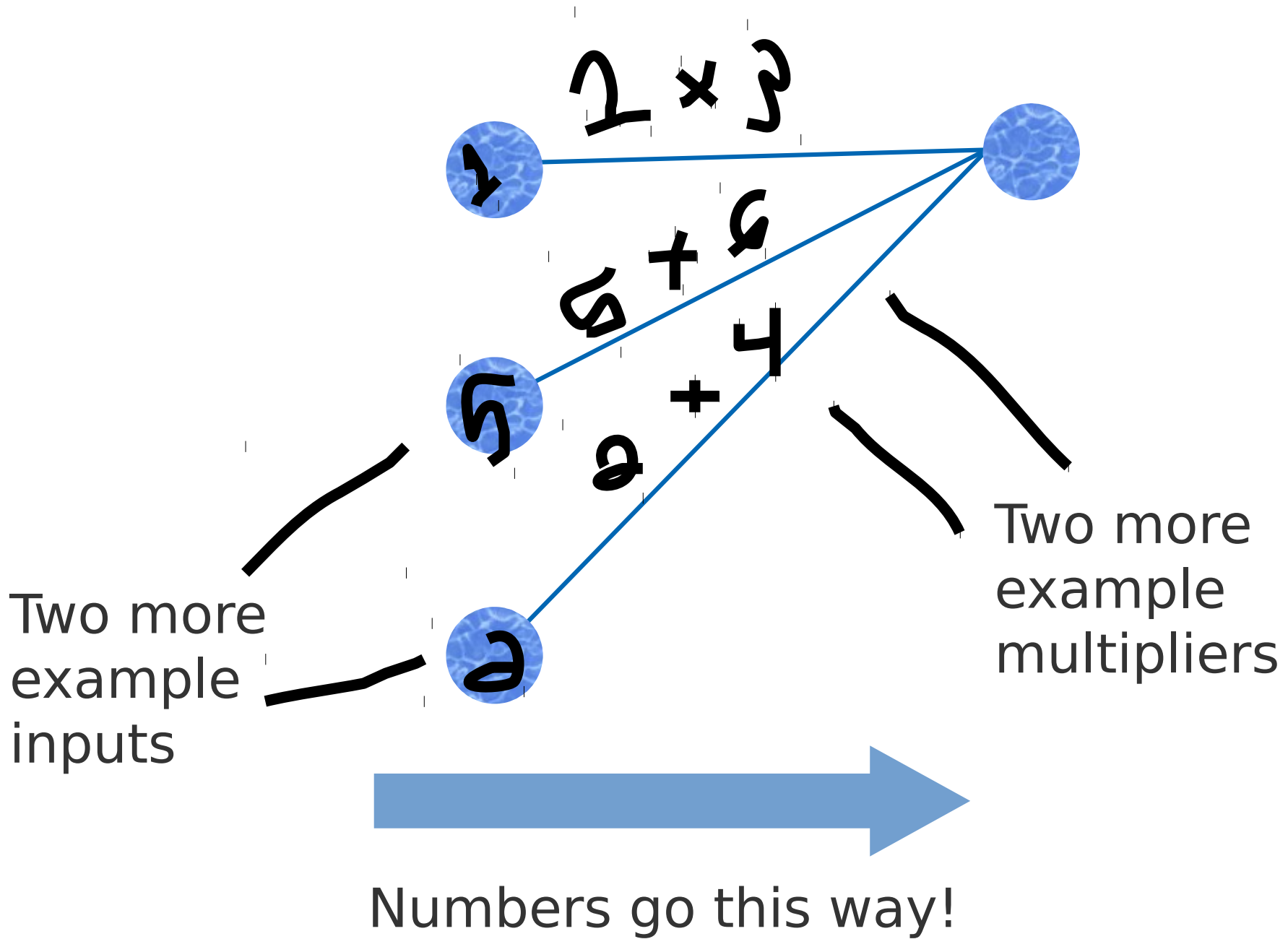


Numbers go this way!

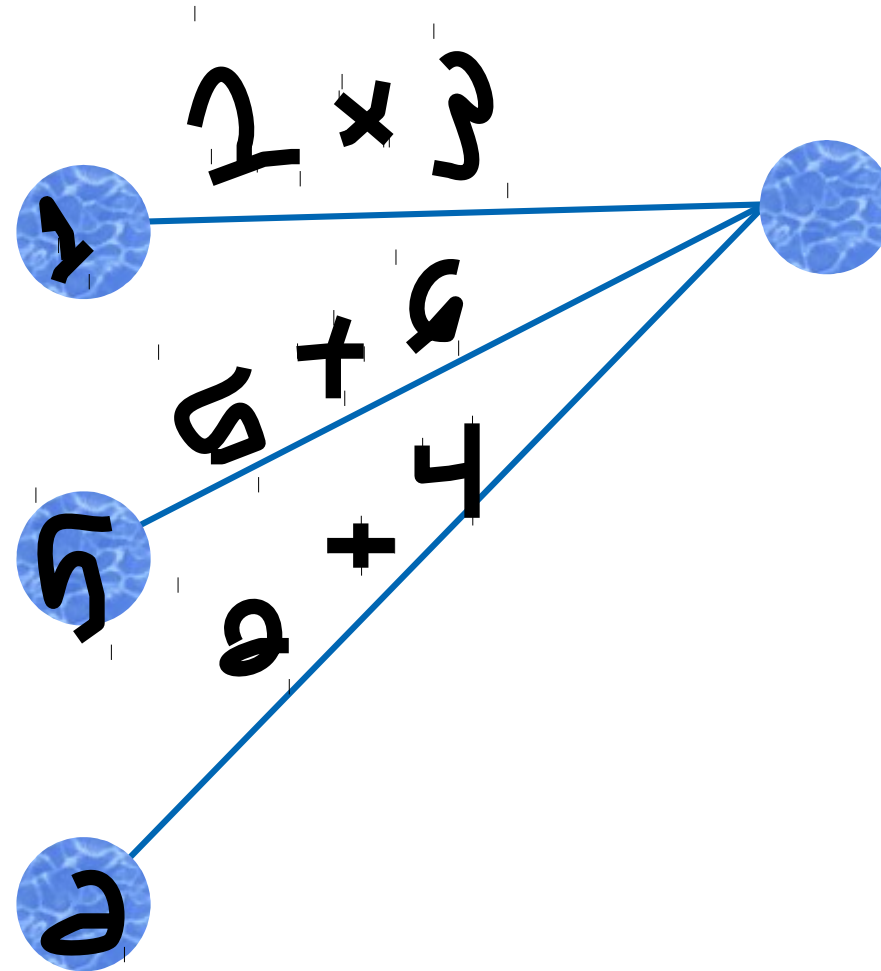
Remember, each line is an input times a multiplier number



Numbers go this way!

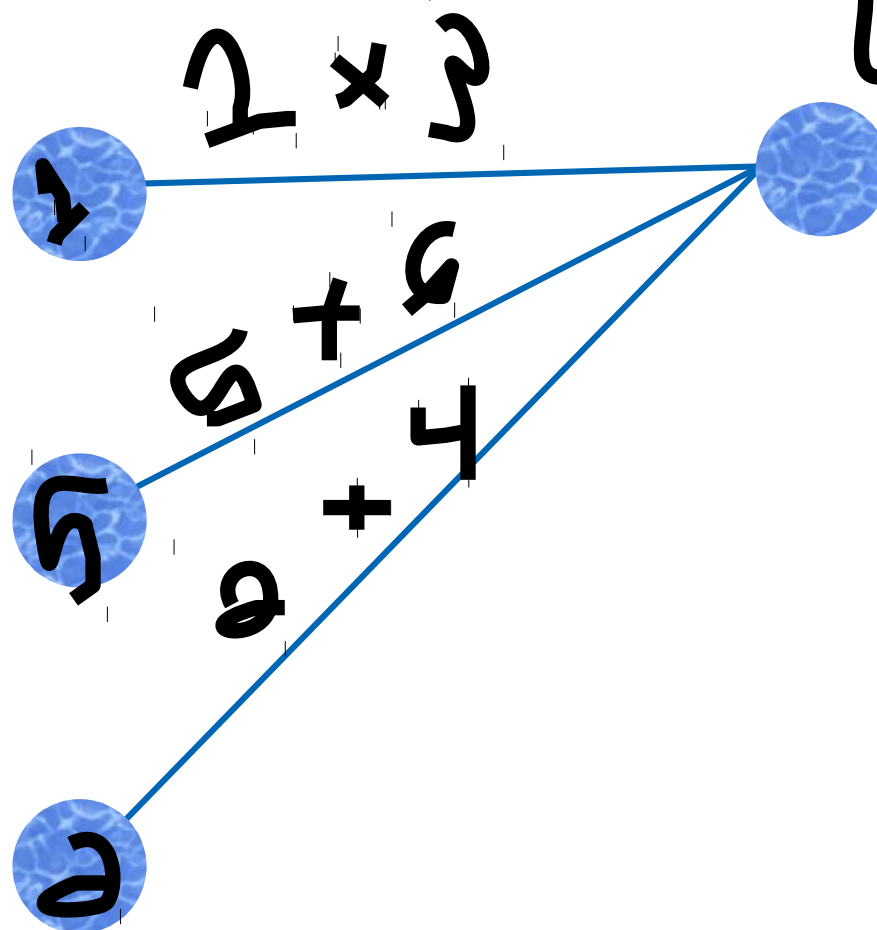


So, all these multiplications flow forward to the middle dot.



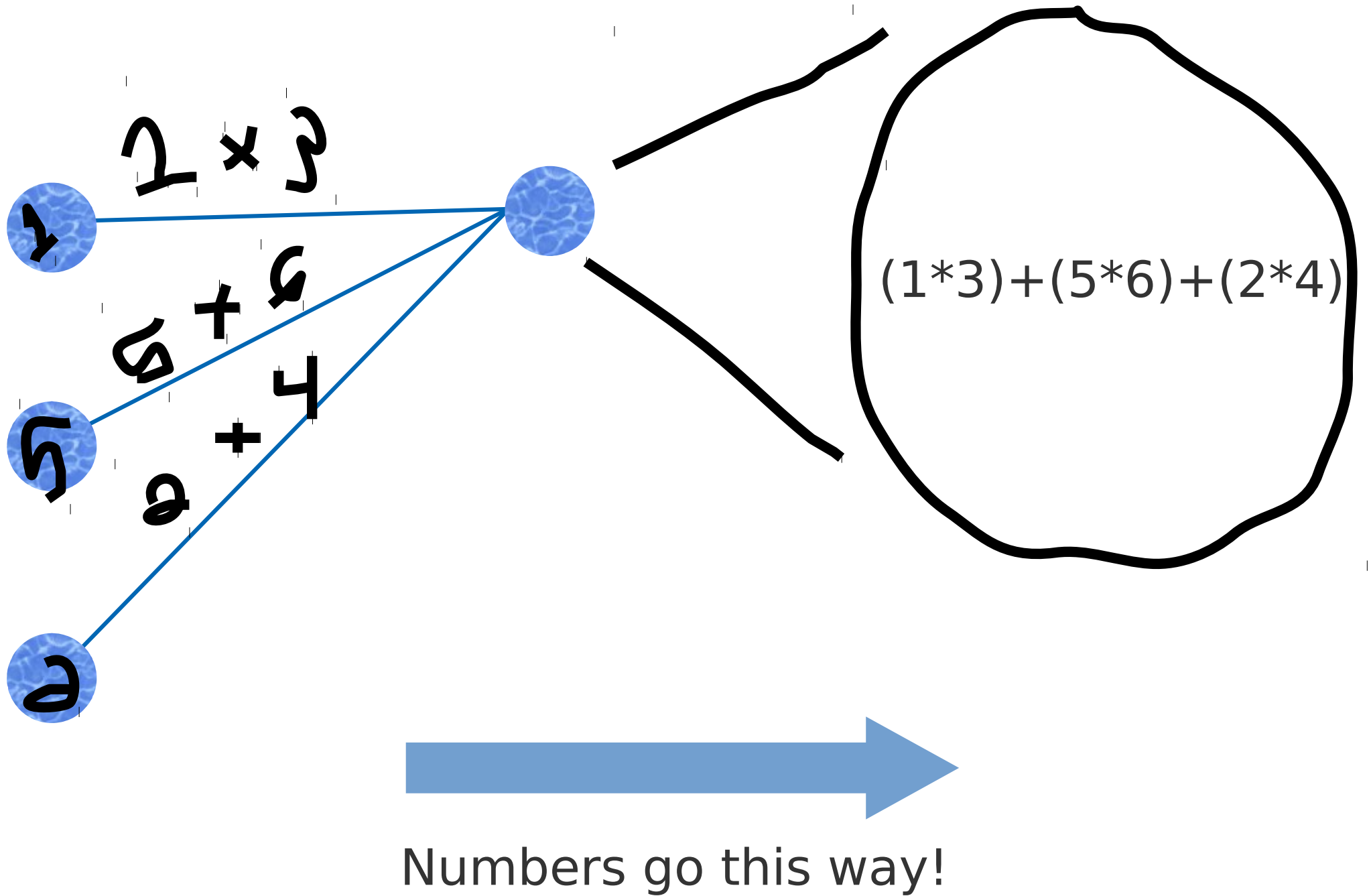
Numbers go this way!

So what happens here?

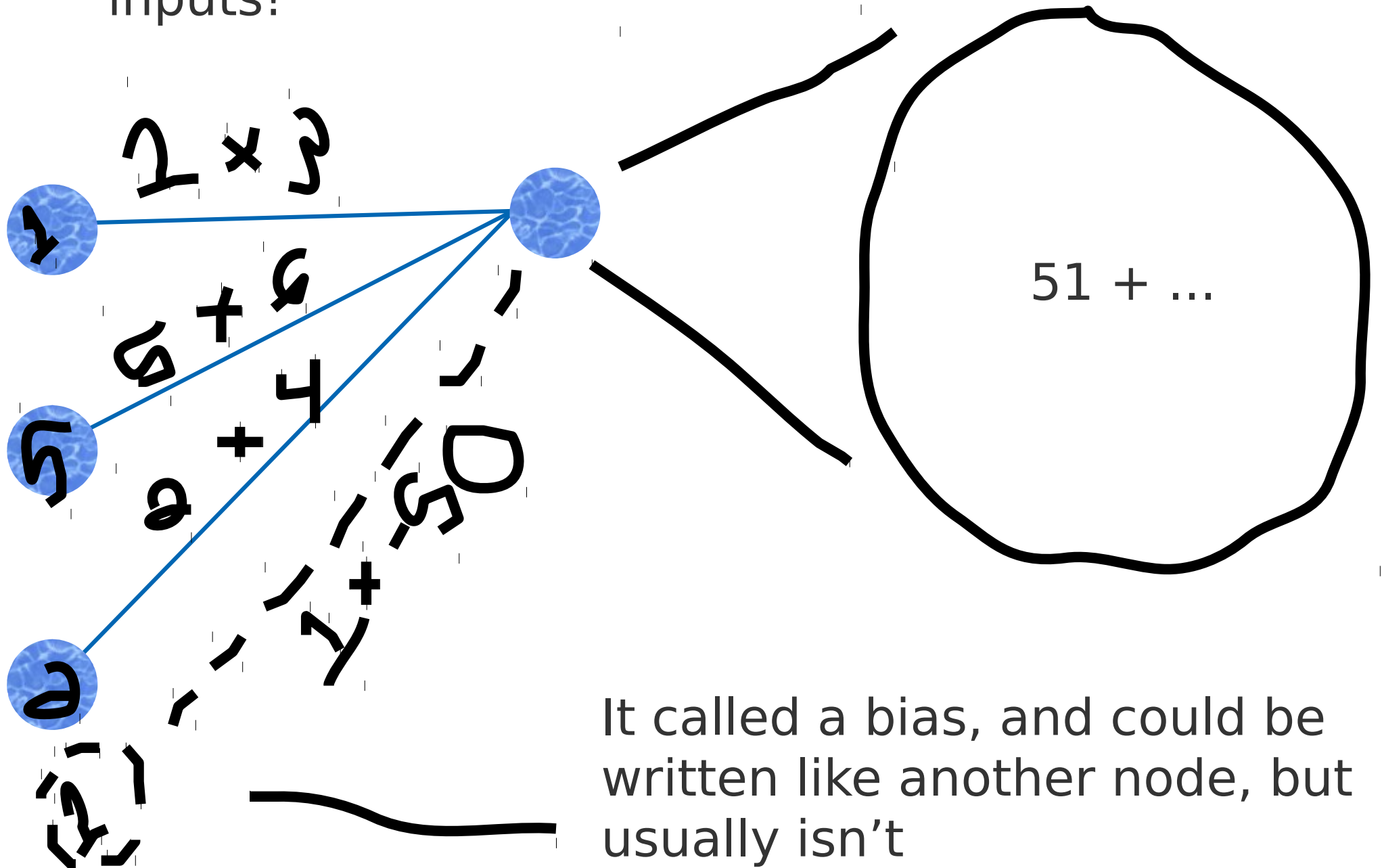


Numbers go this way!

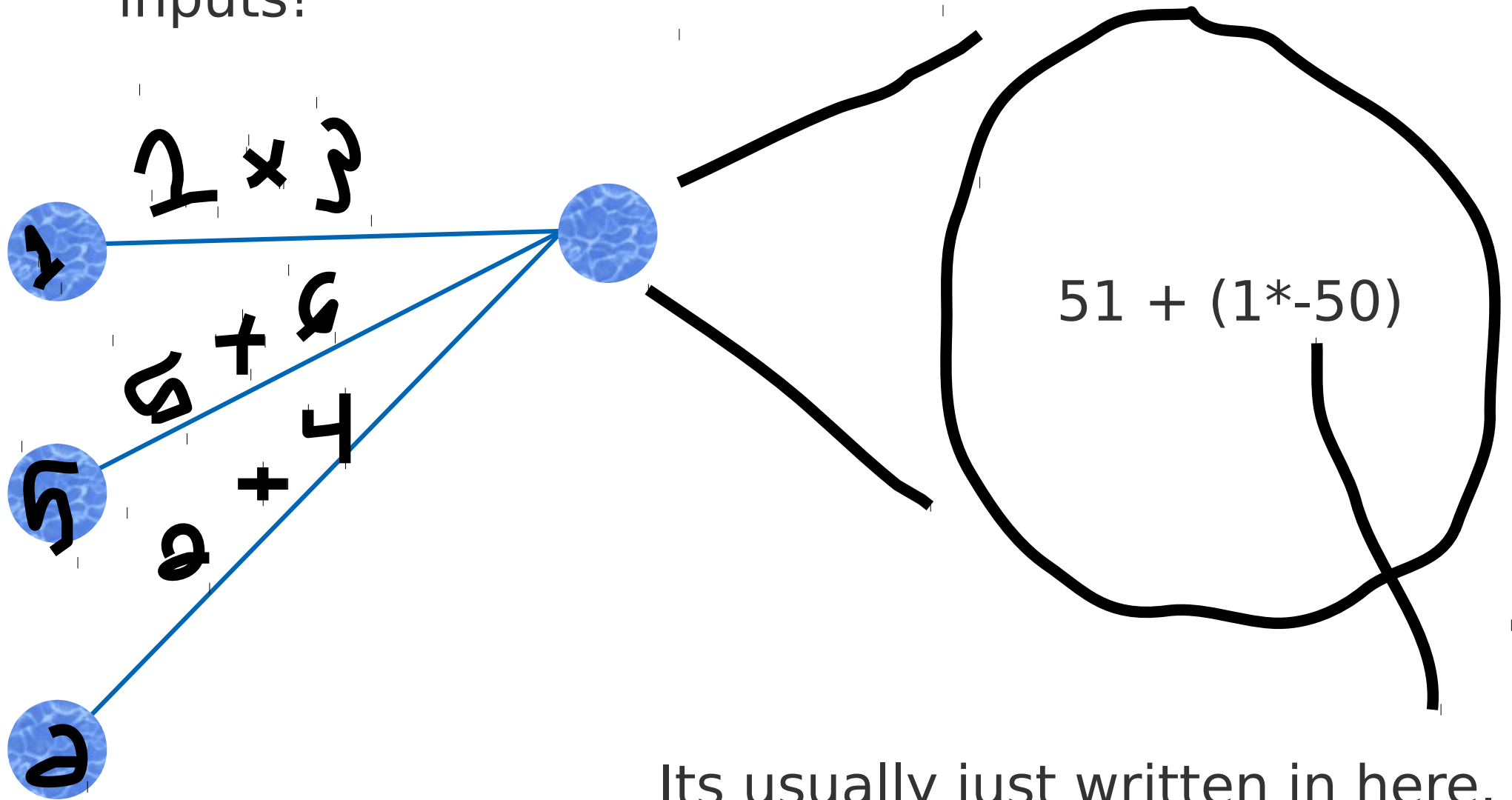
The values are summed up, and ...



We add an extra number independent of the inputs!

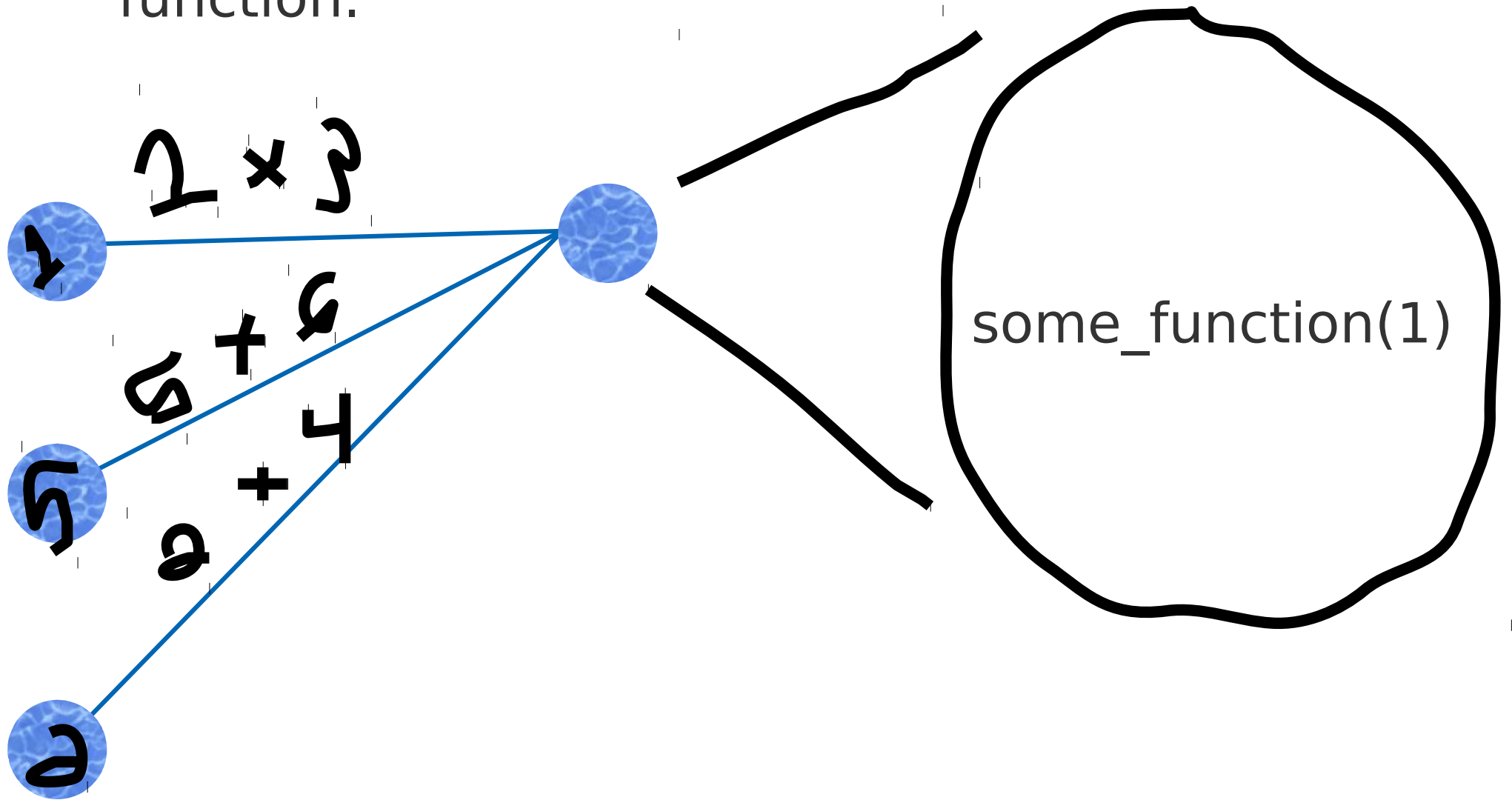


We add an extra number independent of the inputs!

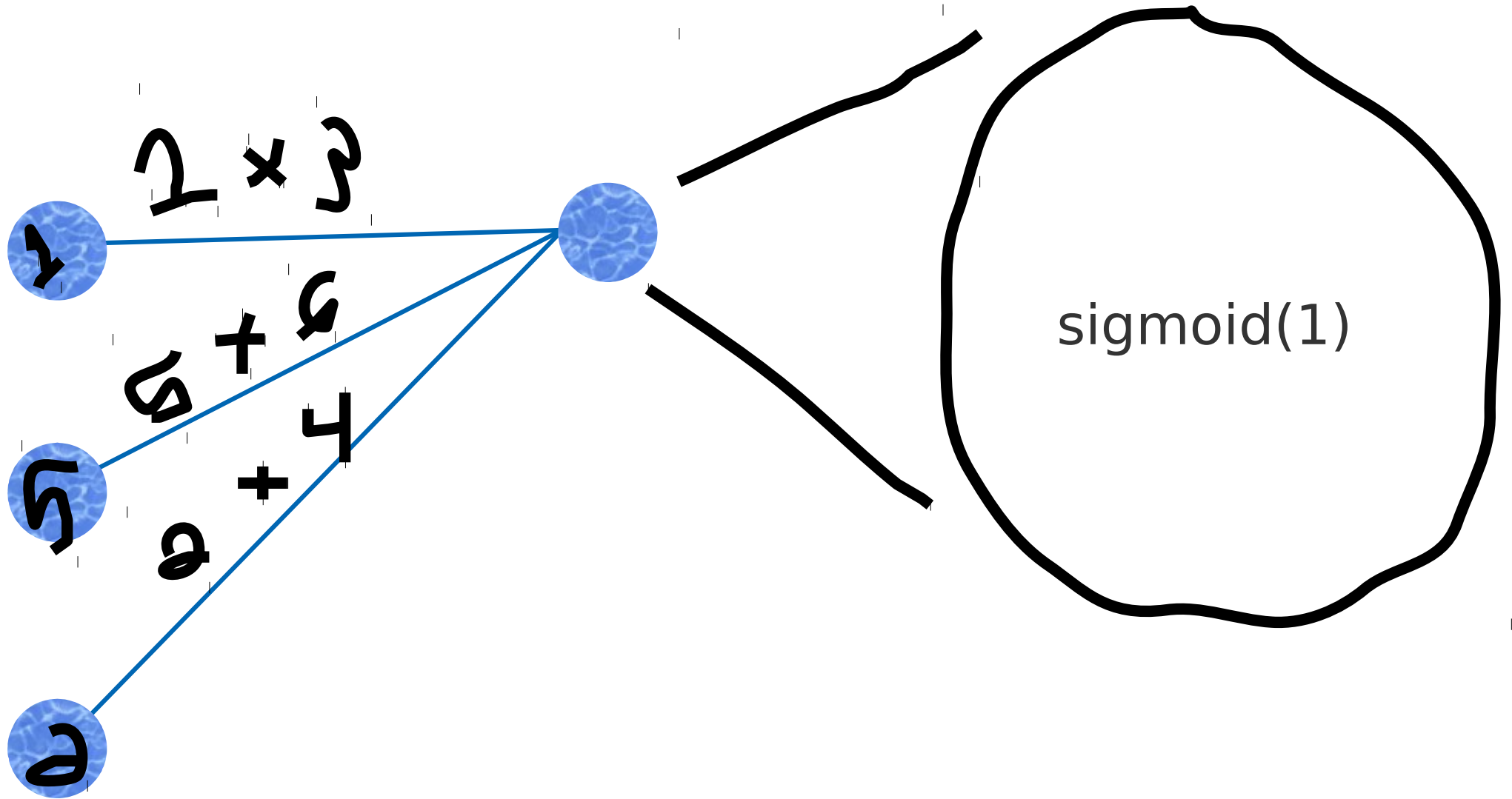


Its usually just written in here.
BUT! It does use a unique multiplier.

Now, the whole result is piped through some function.



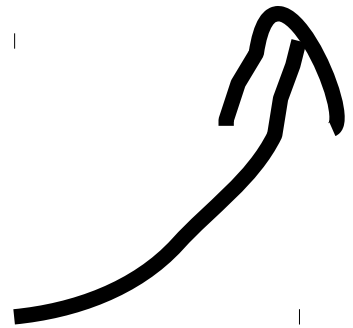
There are many functions that could be used here, one of the common ones is sigmoid



Wait, what is the sigmoid? Its just a function.
Where an “x” goes in and a “y” goes out.

Wait, what is the sigmoid? Its just a function.
Where an “x” goes in and a “y” goes out.

$$y = 1 / (1 + e^{-x})$$

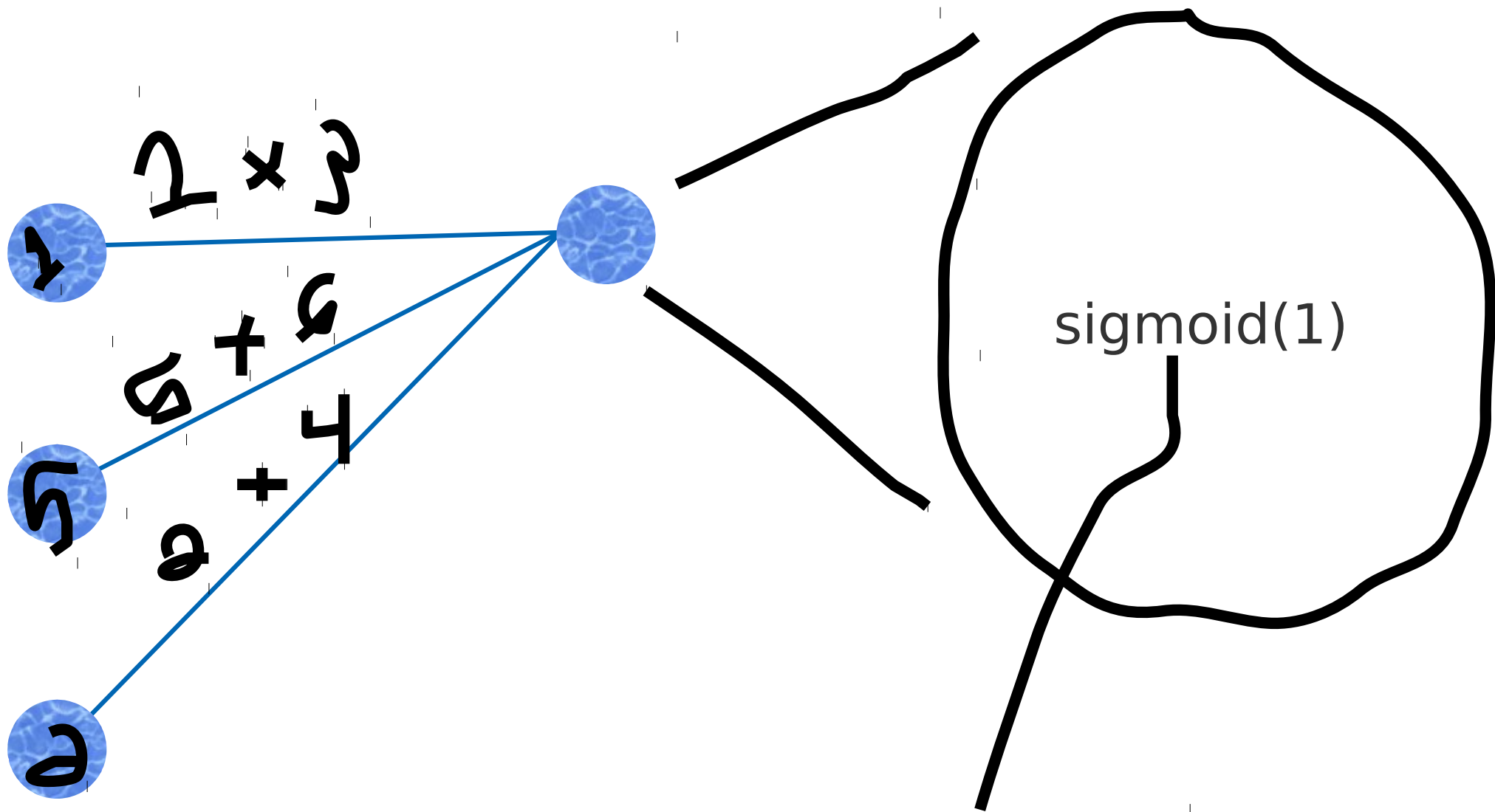


Standard version

The key is, its a **non-linear** function. This gives the neural network the power to model non-linear things.

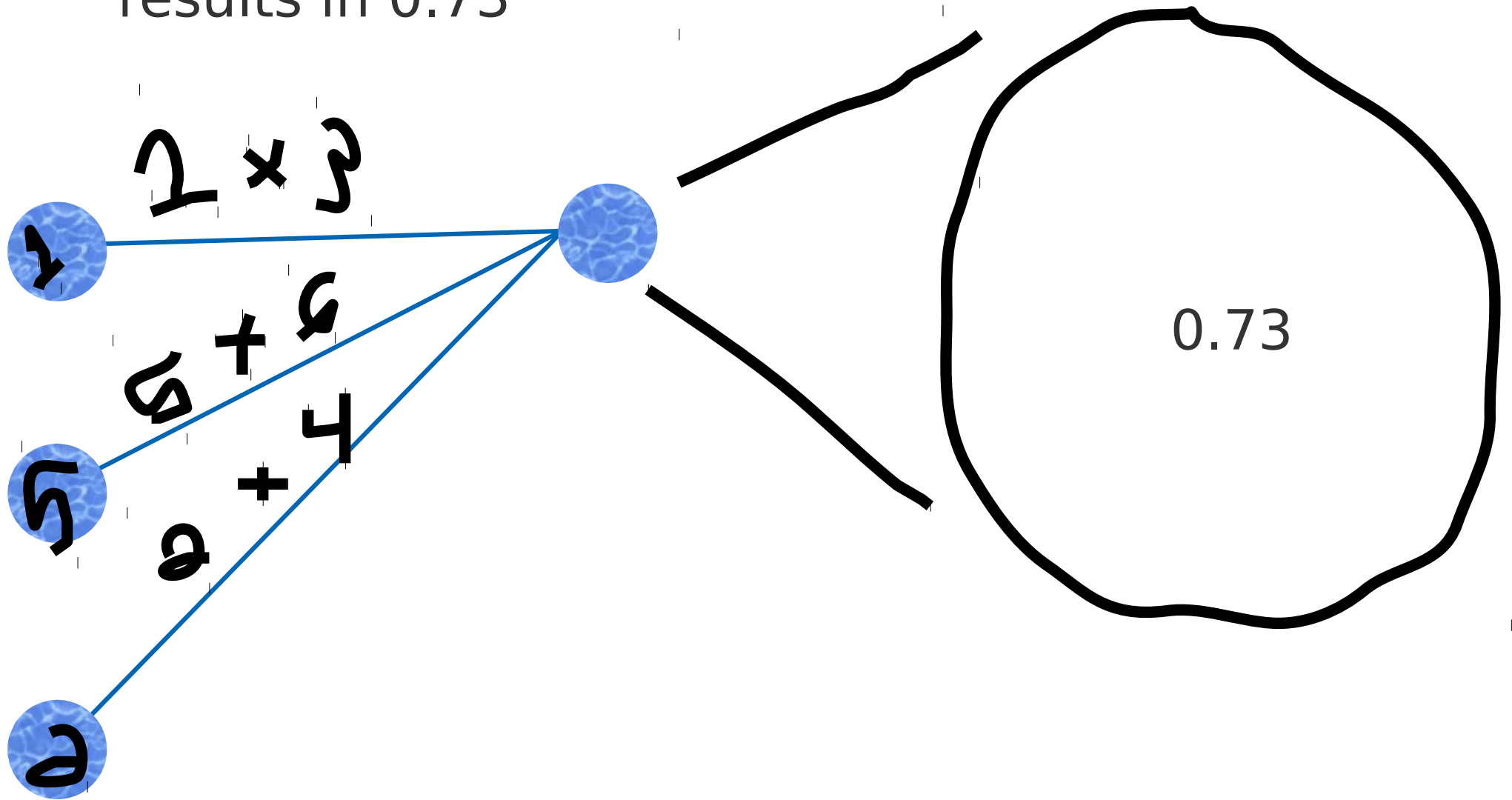
$$y = 1 / (1 + e ^{-x})$$

Back to the graph...

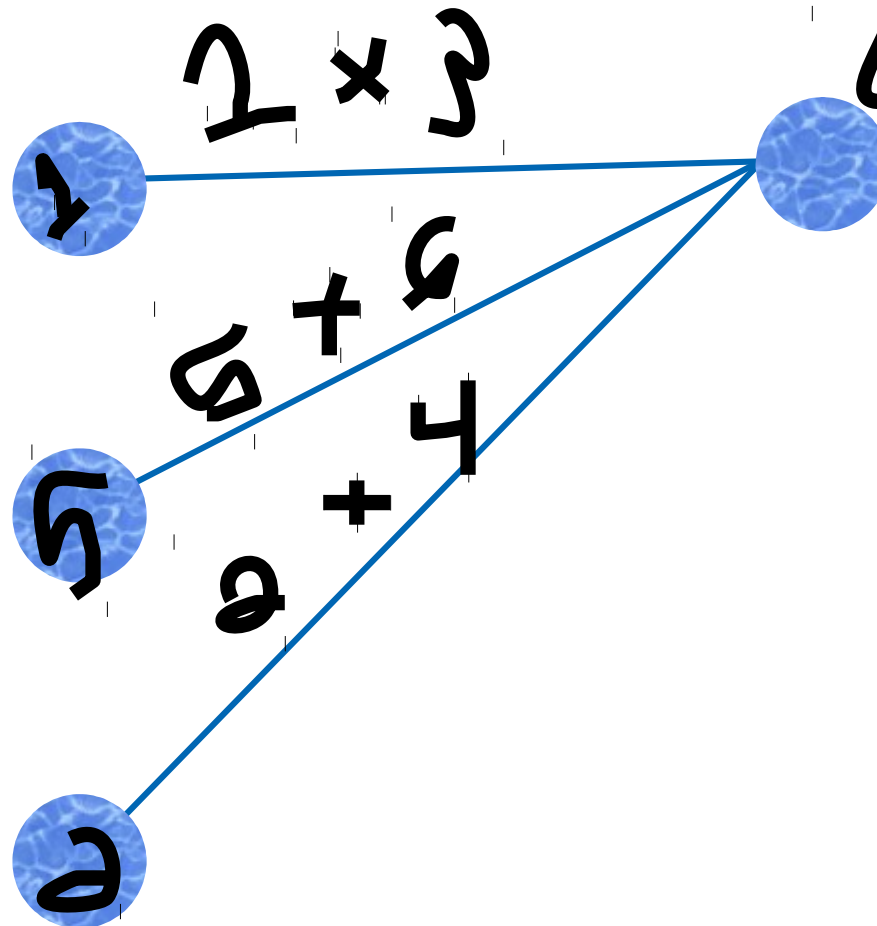


Here's the sigmoid, in code-like form

The sigmoid min is -1 and max is 1. This one results in 0.73

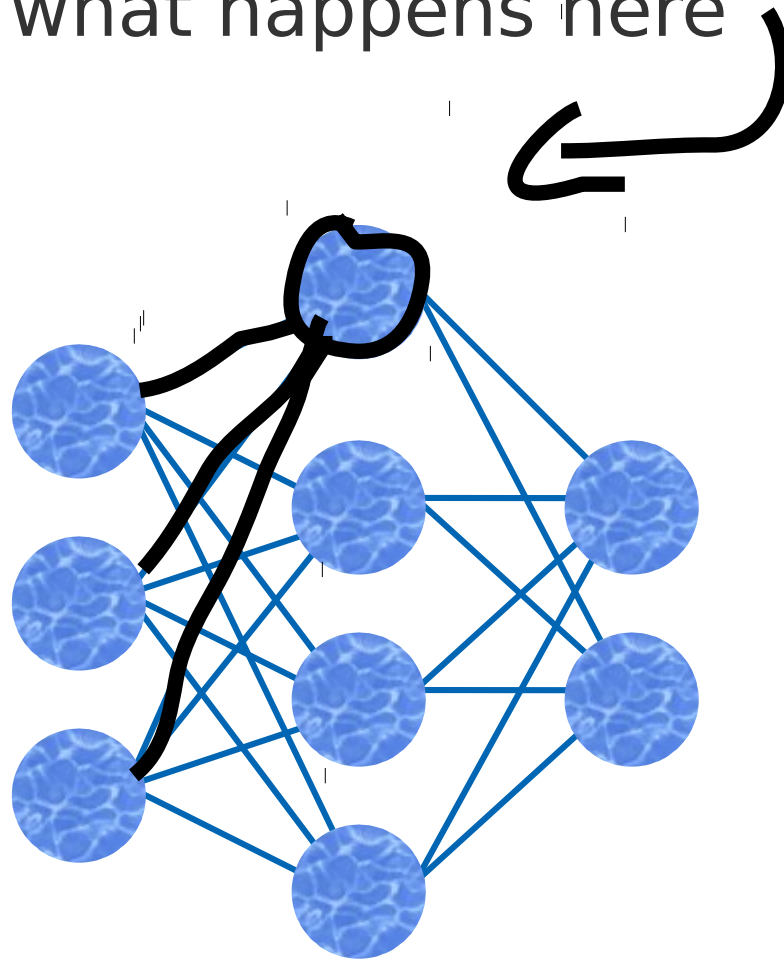


So, the value here is 0.73



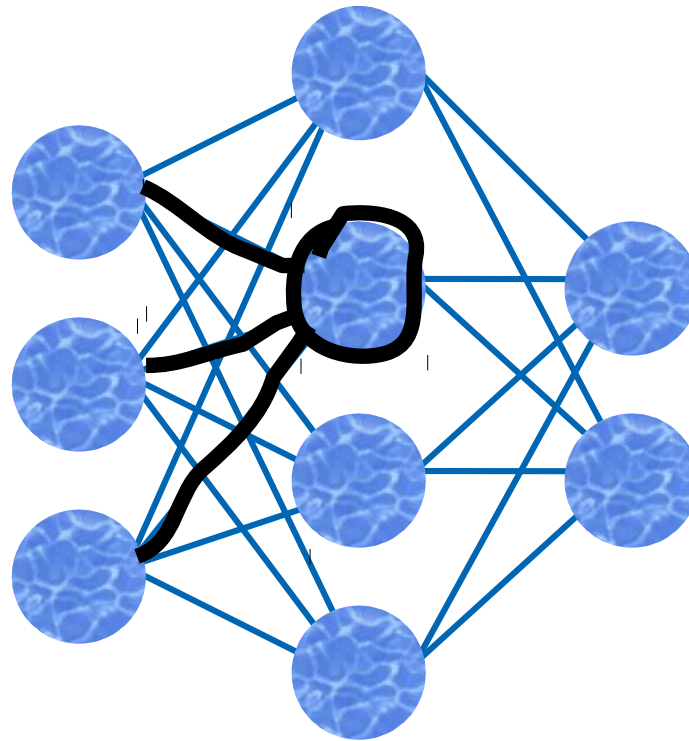
Numbers go this way!

We just saw what happens here



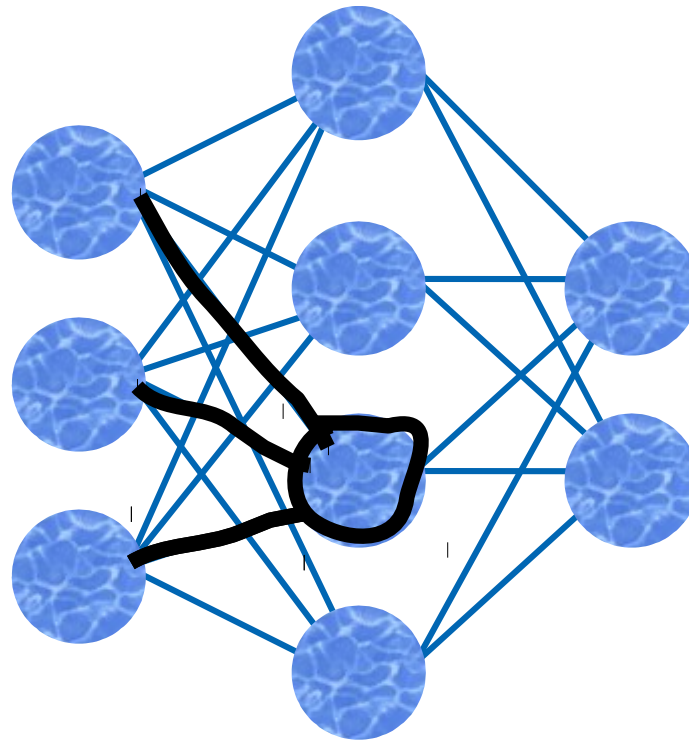
Numbers go this way!

This process is repeated with unique multipliers here...



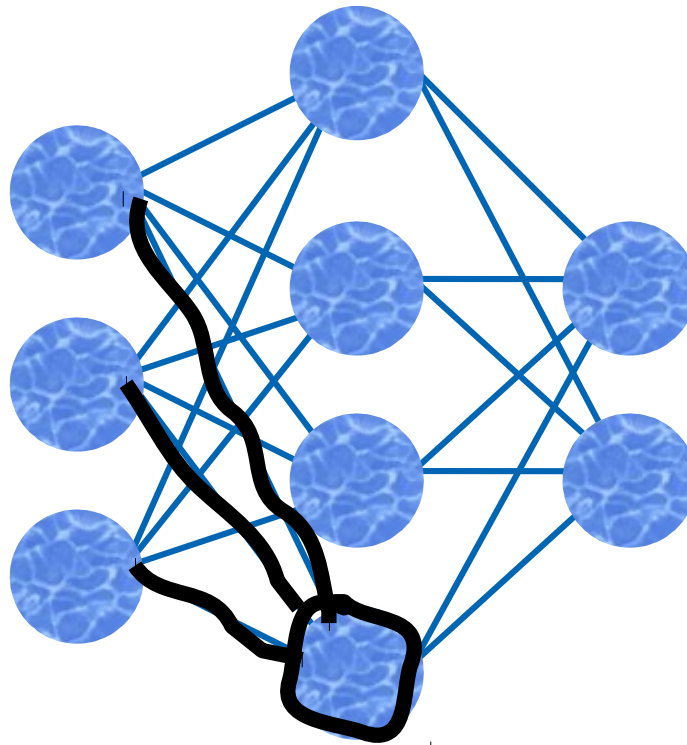
Numbers go this way!

here...



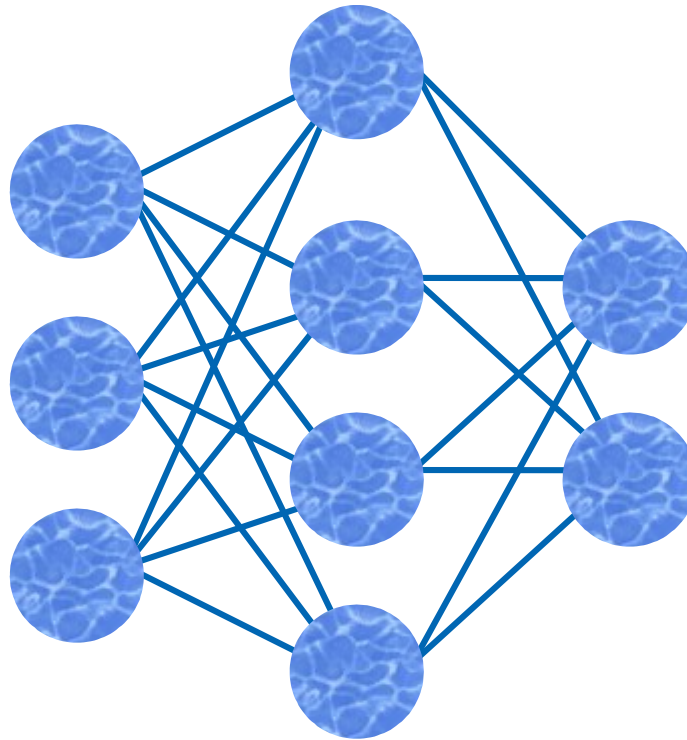
Numbers go this way!

...and here.



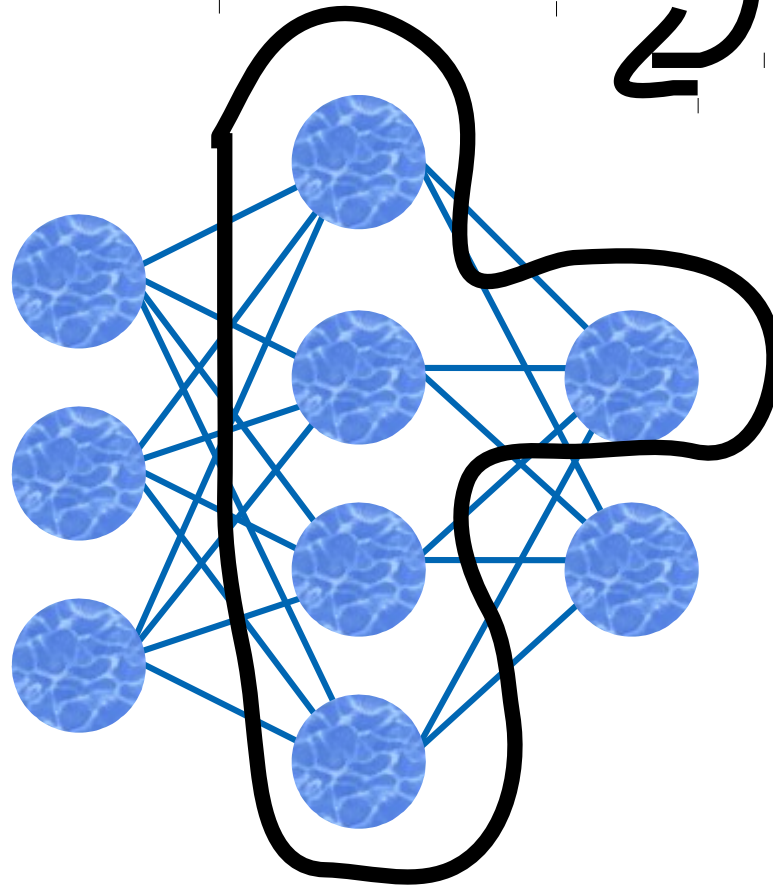
Numbers go this way!

What happens next?



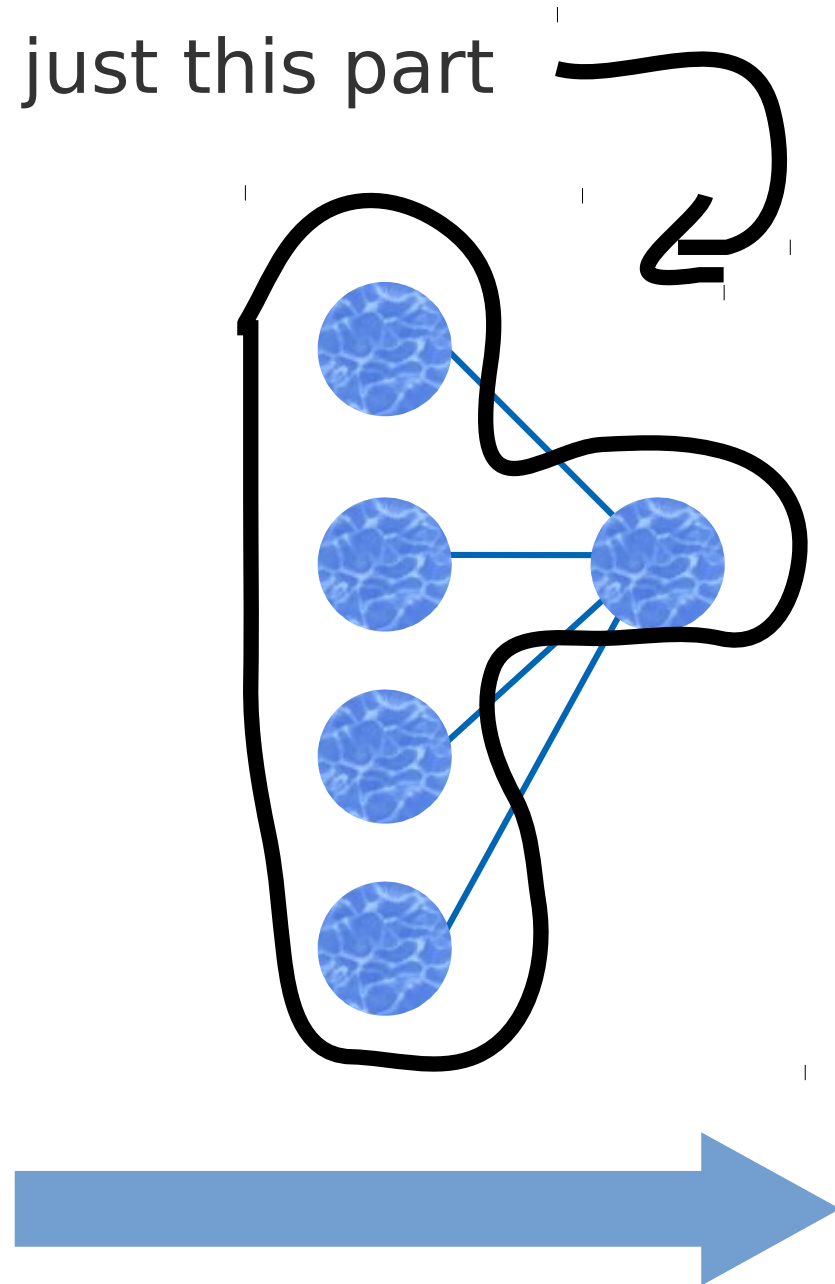
Numbers go this way!

Lets look at just this part



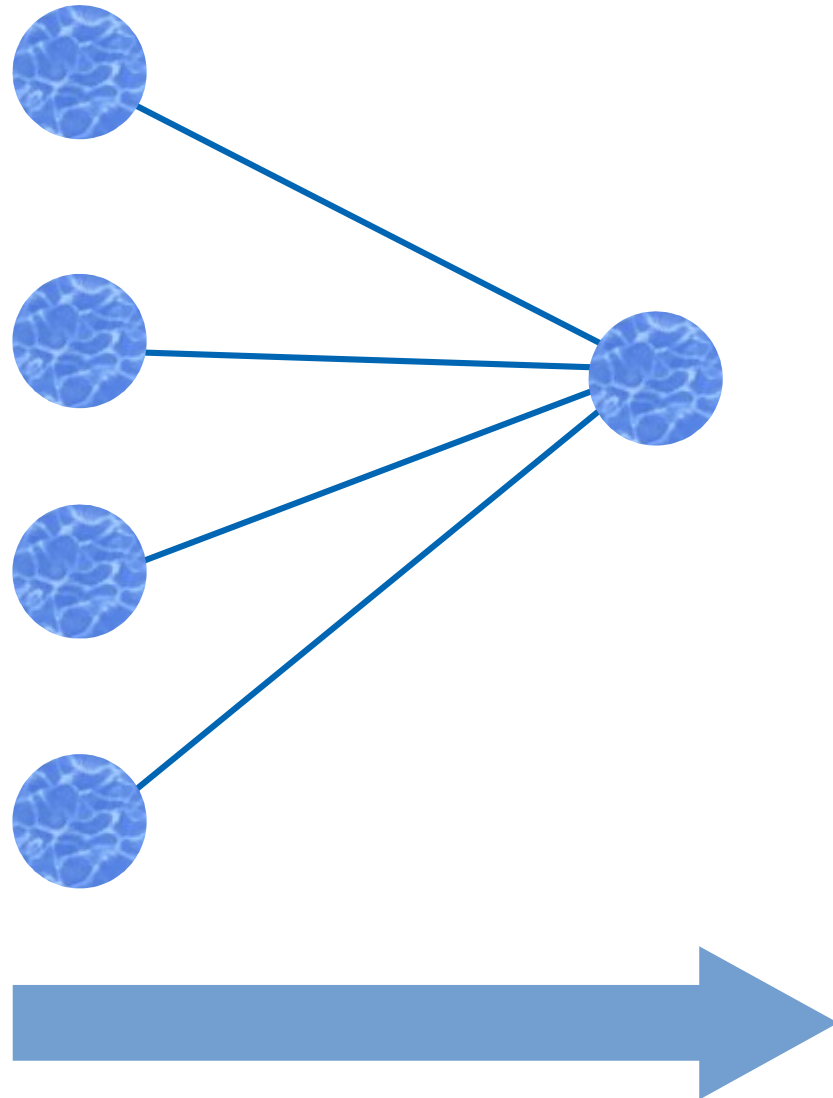
Numbers go this way!

Lets look at just this part



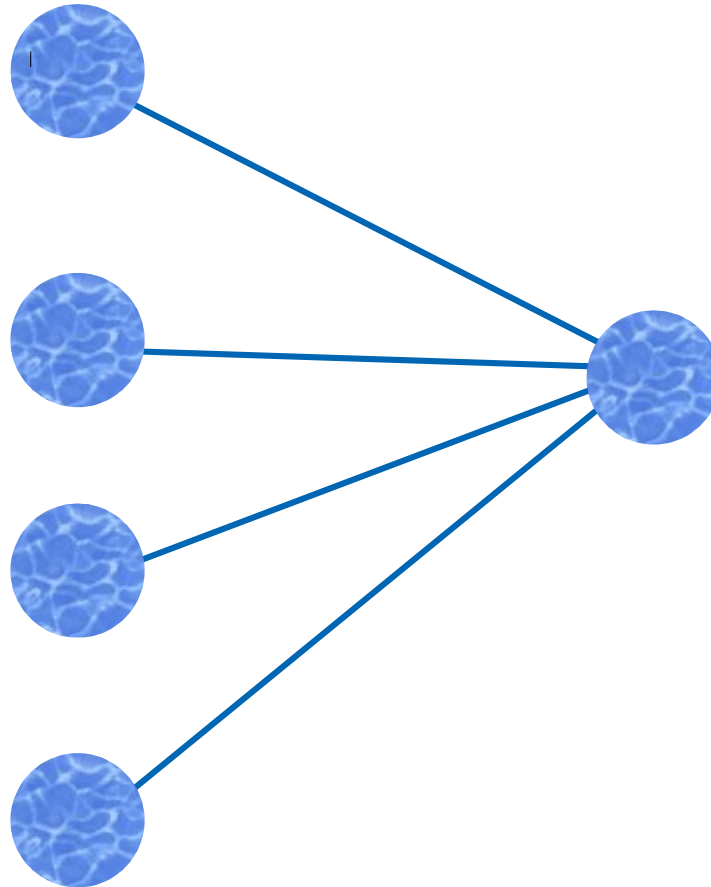
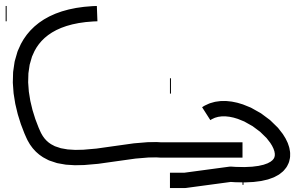
Numbers go this way!

Spread them out a tad



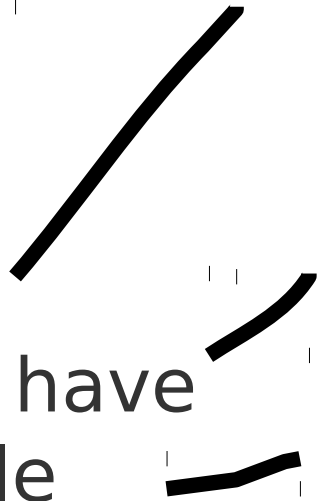
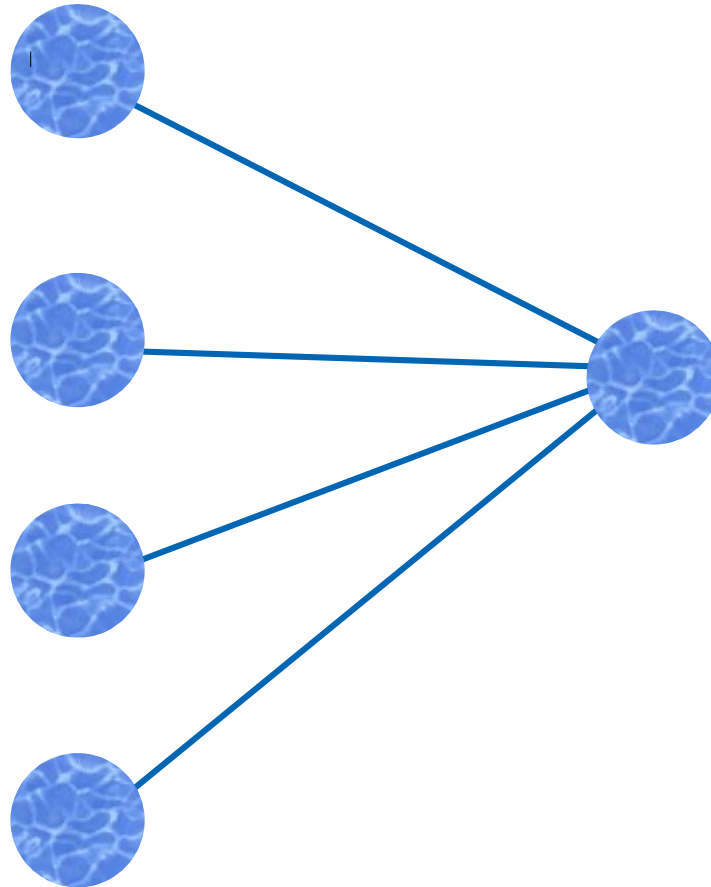
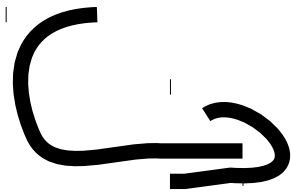
Numbers go this way!

Remember this was 0.73



Numbers go this way!

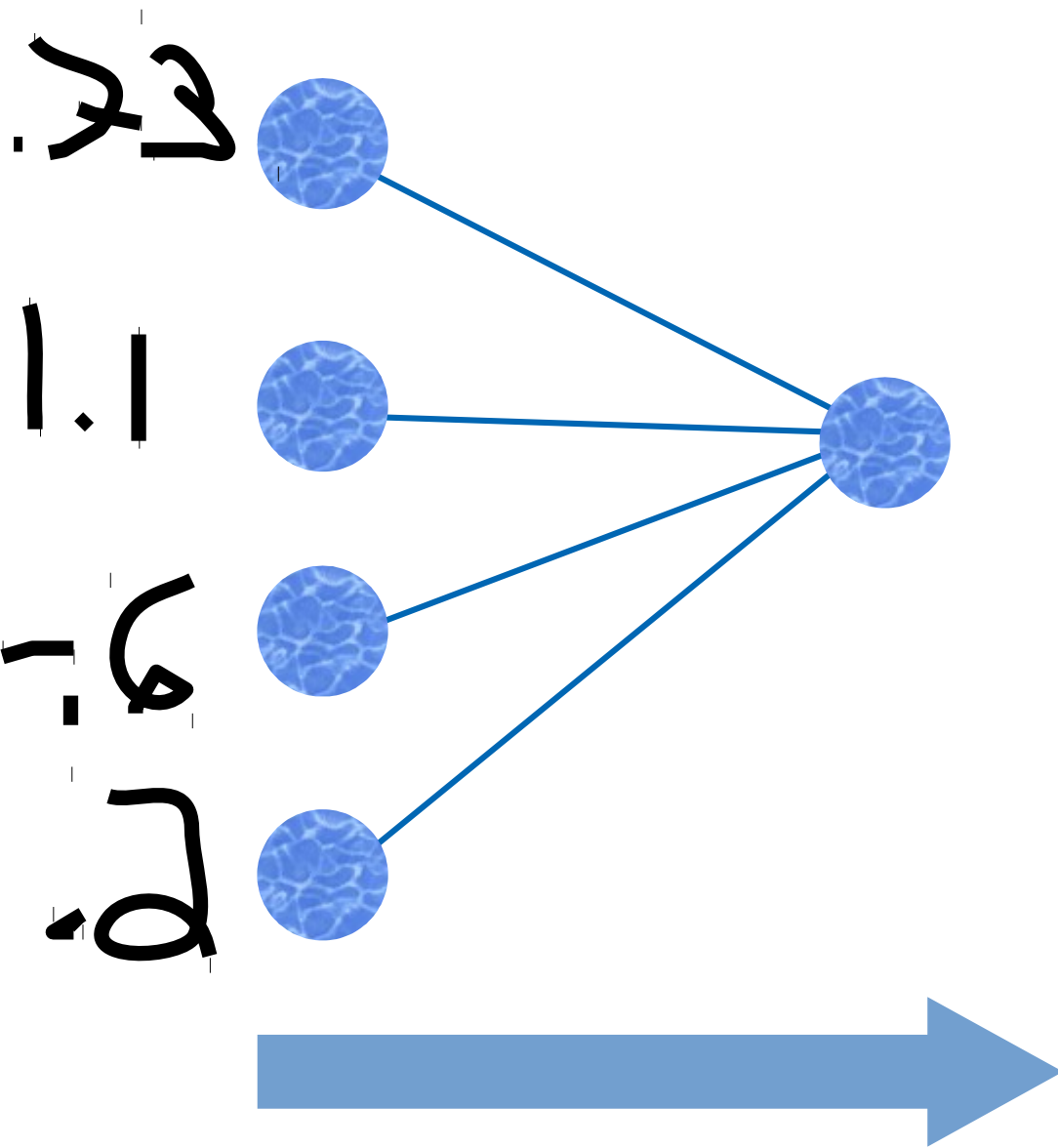
Remember this was 0.73



These have
a single
value also

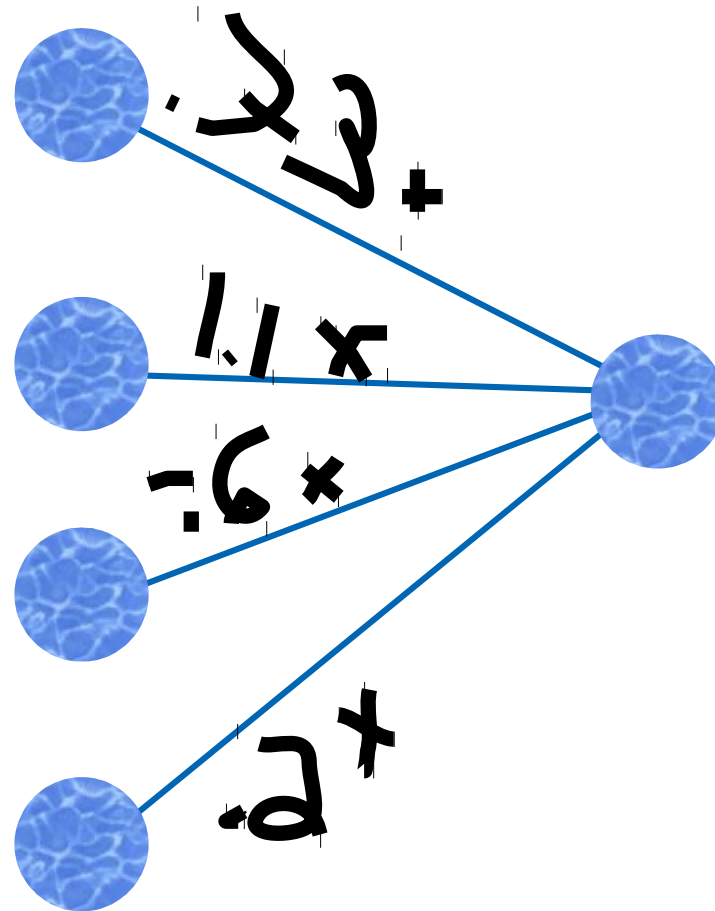


Numbers go this way!



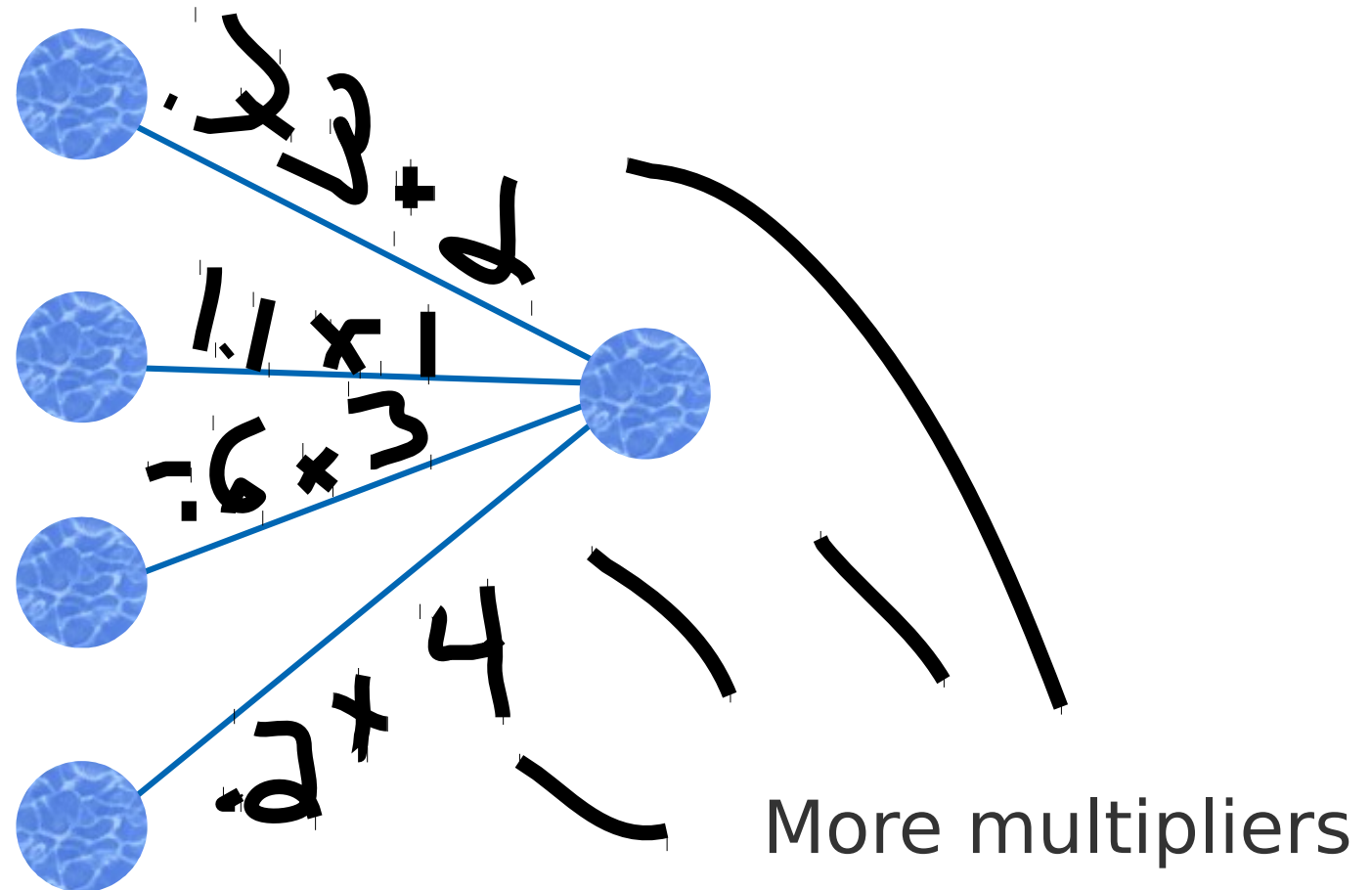
Numbers go this way!

By now maybe you know the drill...



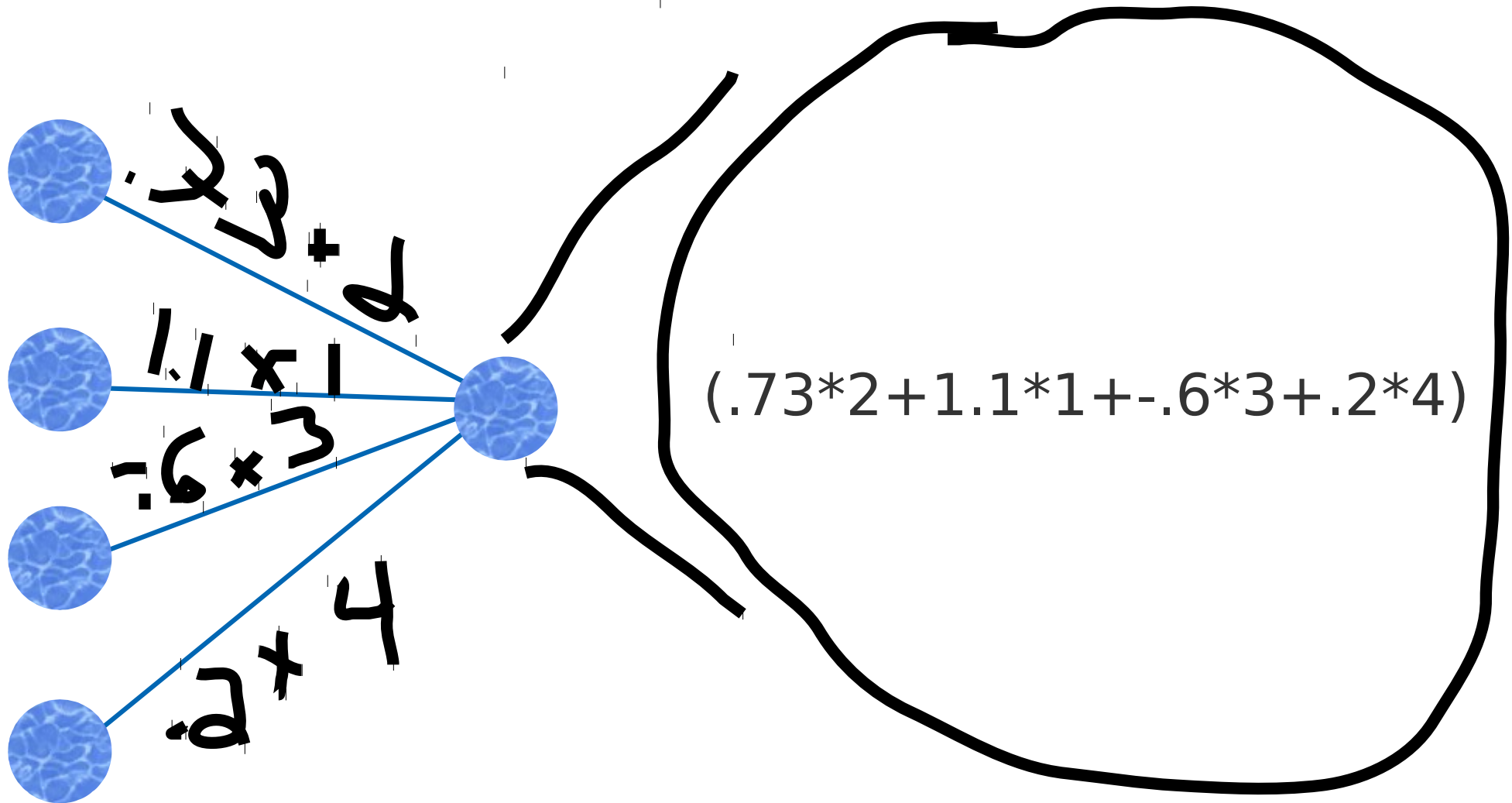
Numbers go this way!

By now maybe you know the drill...



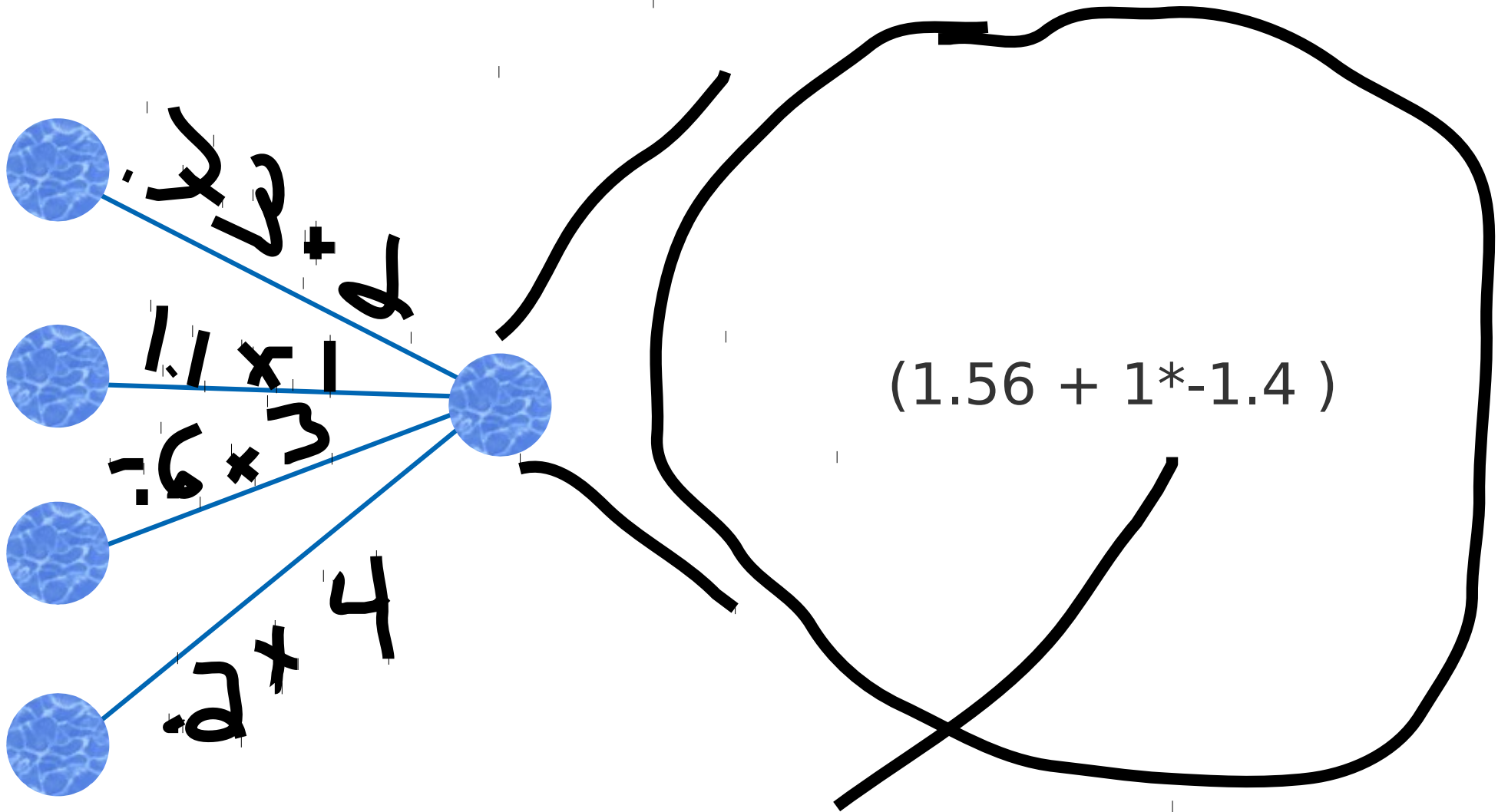
Numbers go this way!

By now maybe you know the drill...



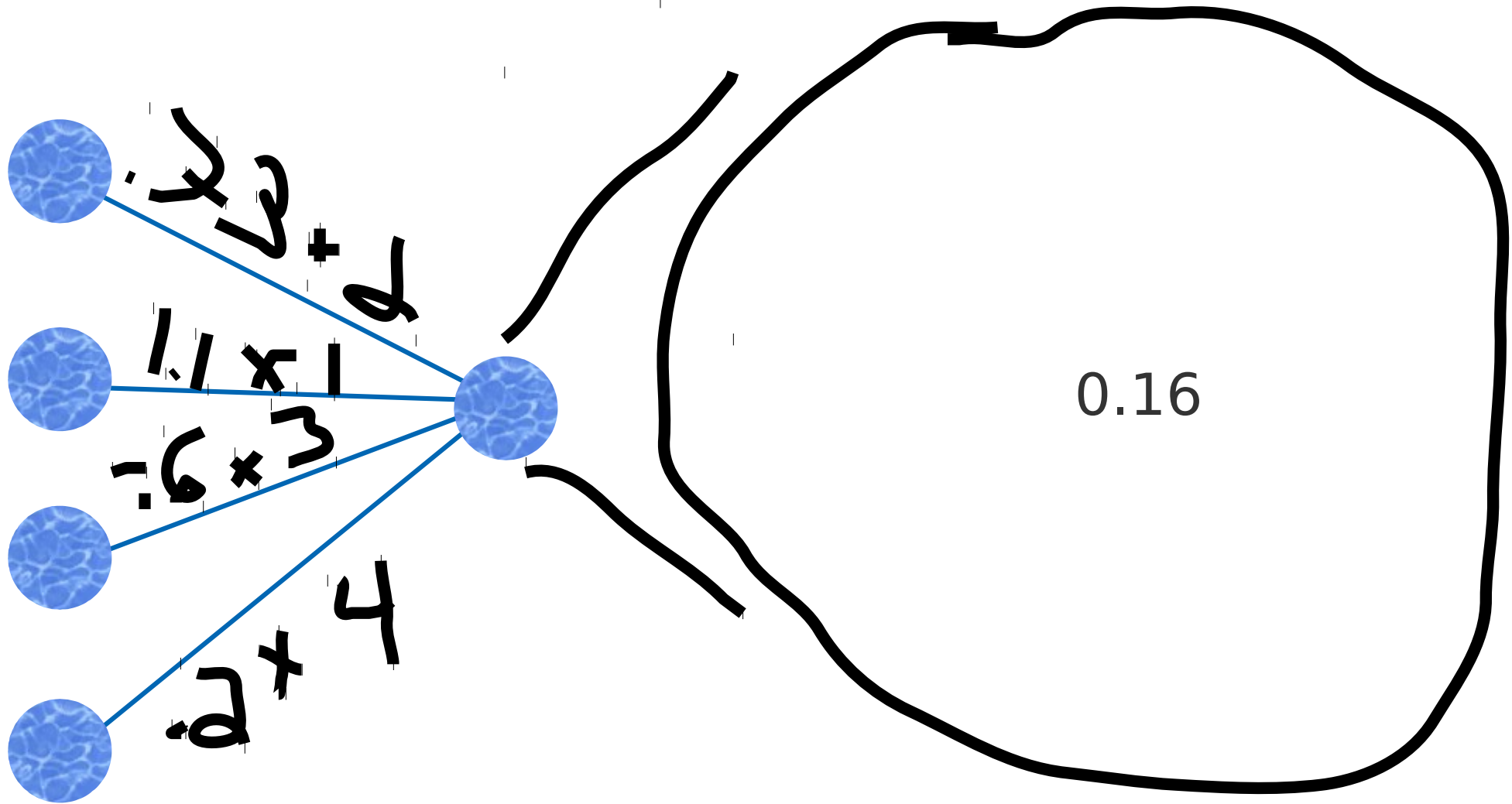
Numbers go this way!

By now maybe you know the drill...



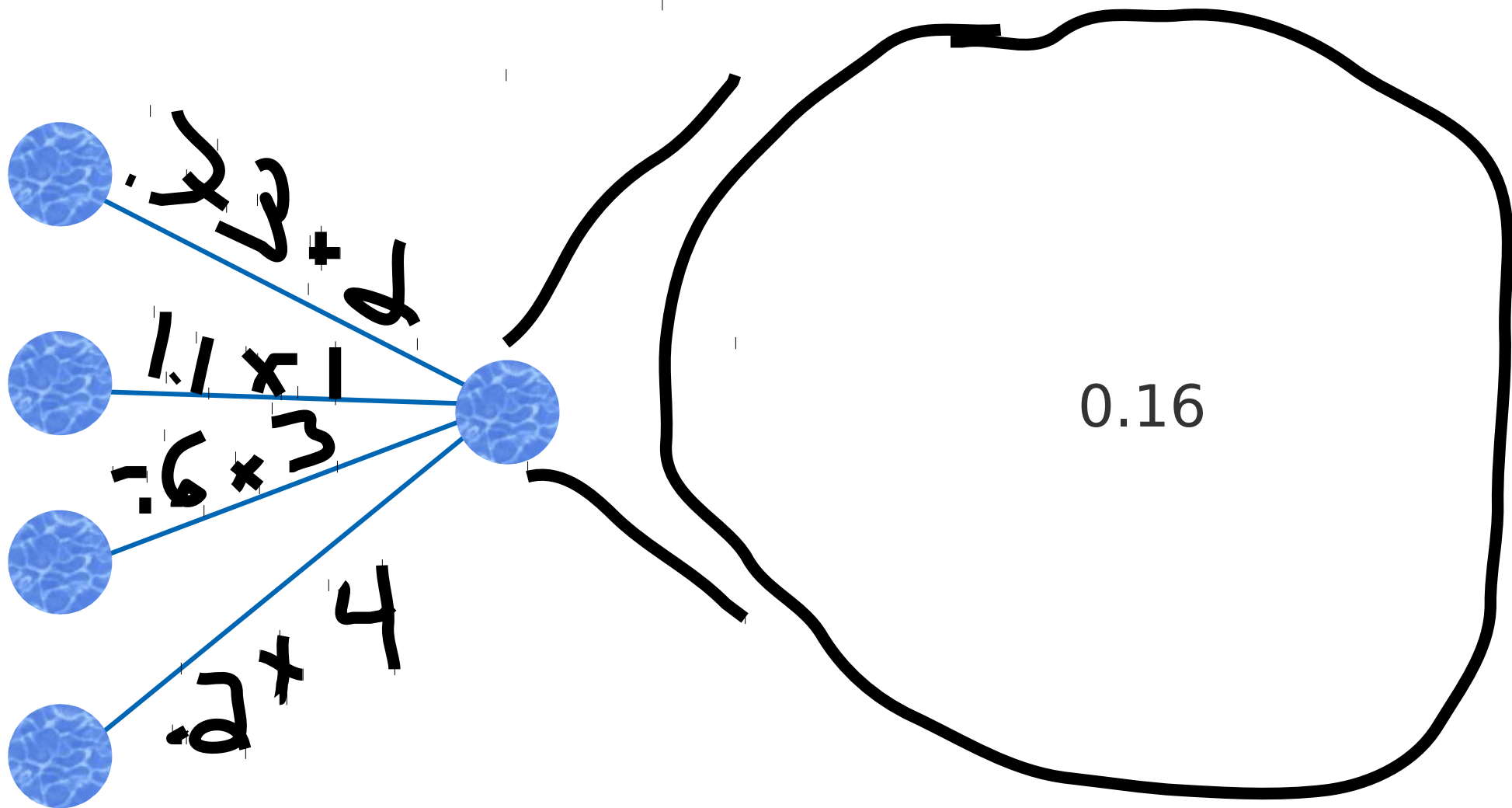
Remember we sneak
in the bias here?

By now maybe you know the drill...



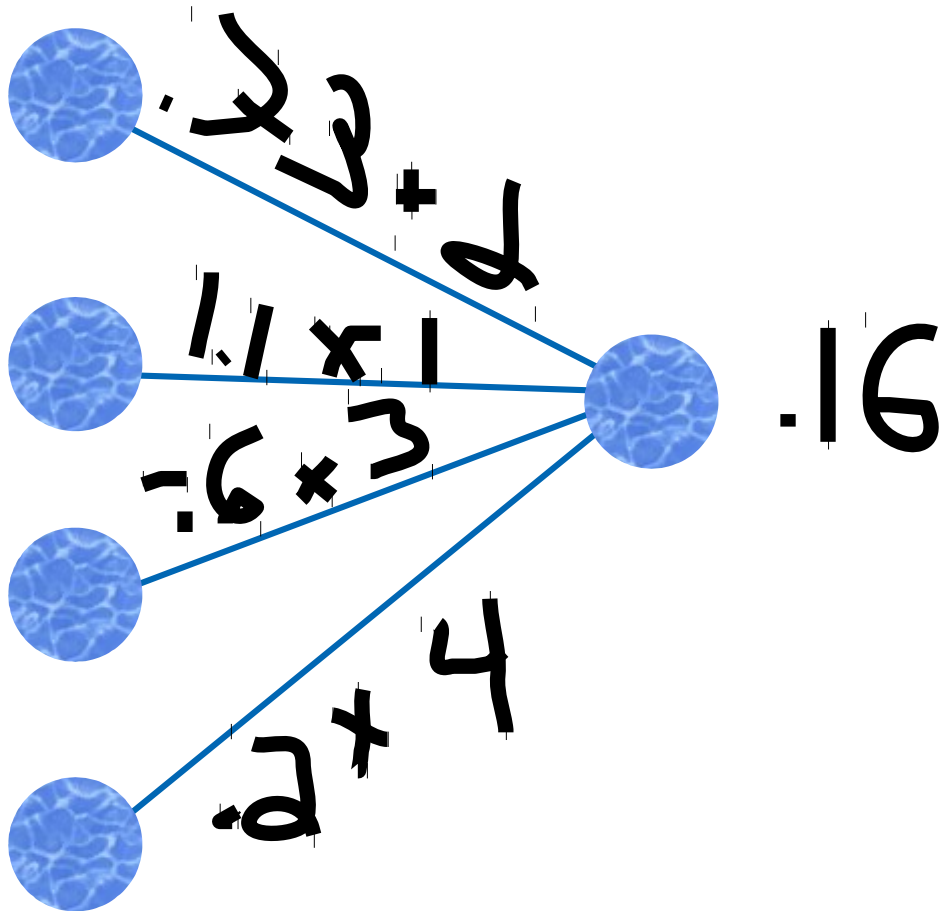
Numbers go this way!

Now we do the activation... **WAIT!**



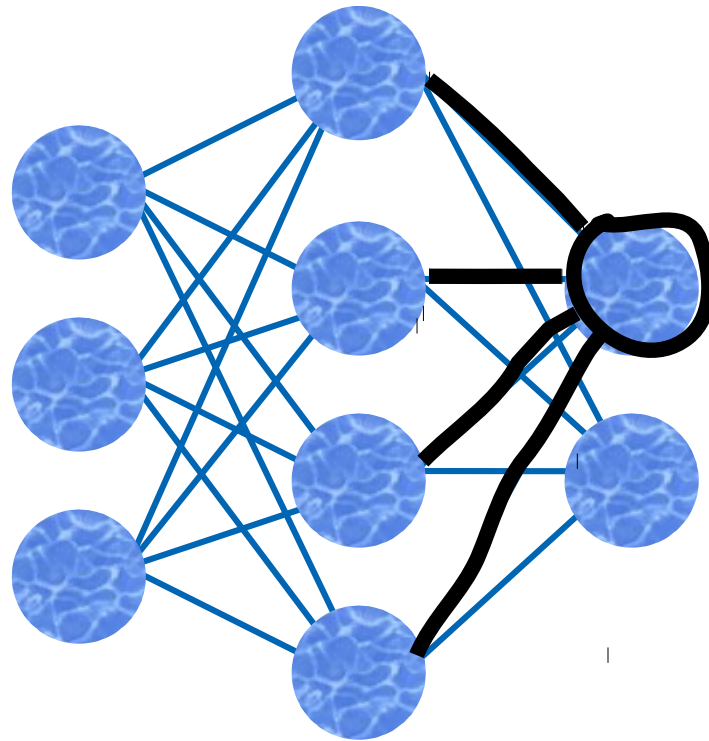
Numbers go this way!

We don't do the activation in the final layer. We will do something else instead at the very end.



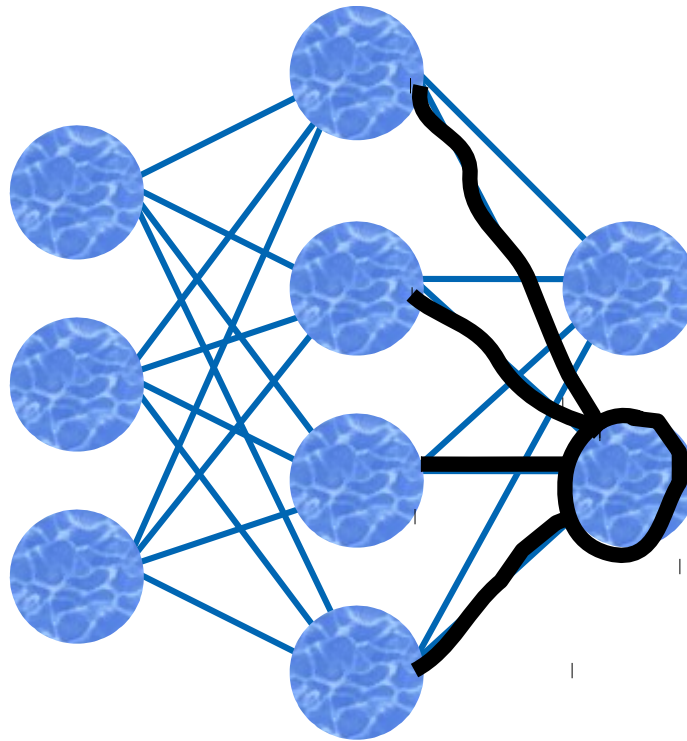
Numbers go this way!

We just looked at what happens here.



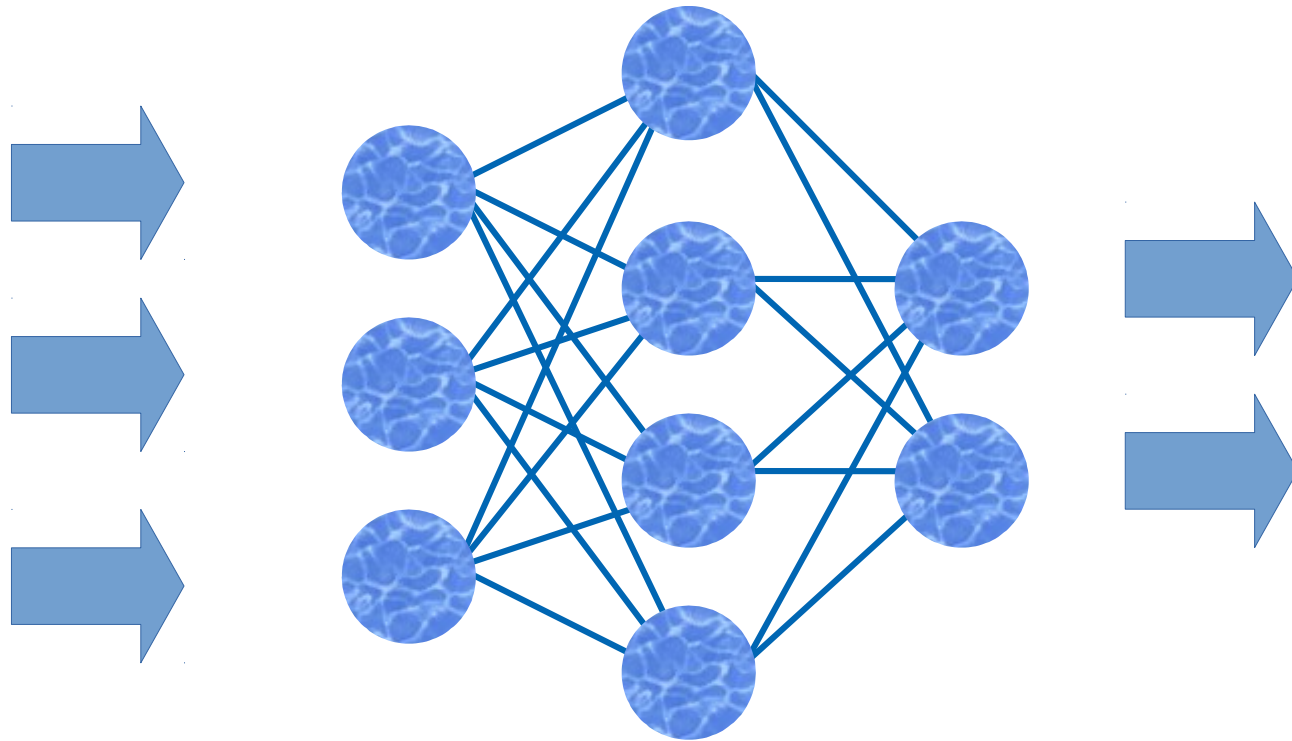
Numbers go this way!

We repeat that process here with more multipliers.



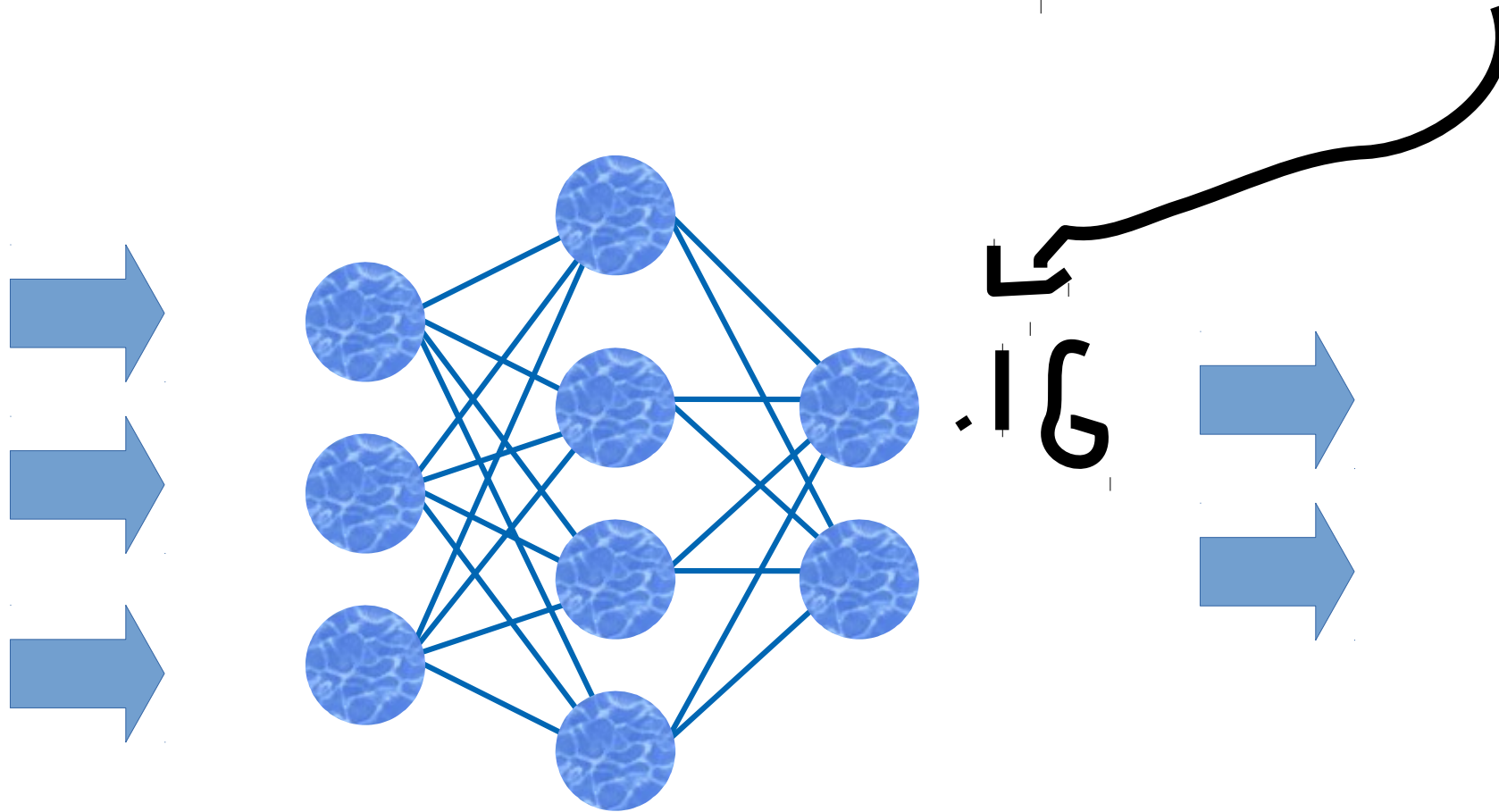
Numbers go this way!

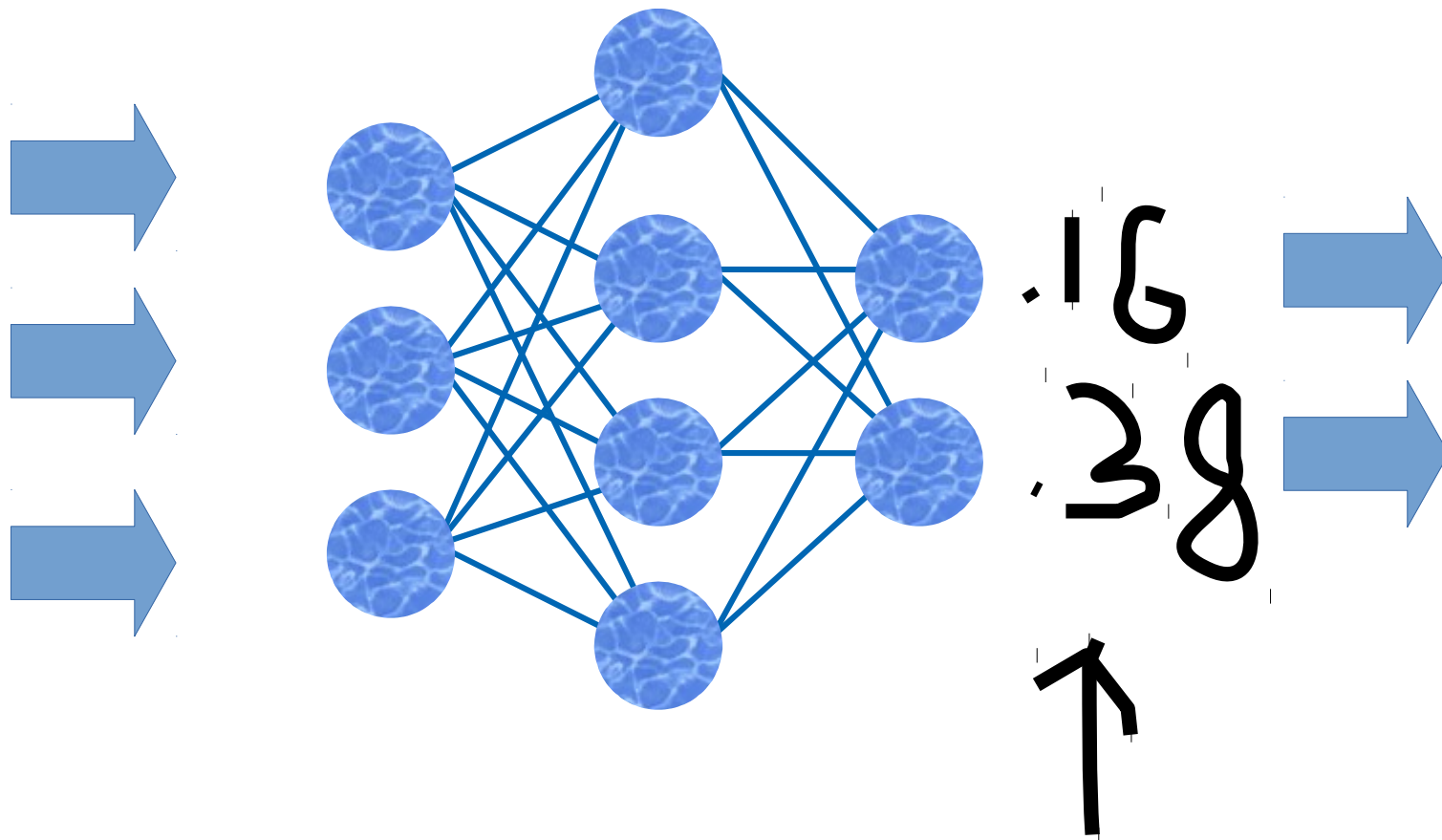
Ok, three numbers went in, and two numbers came out.



What are we going to do with the last two numbers?

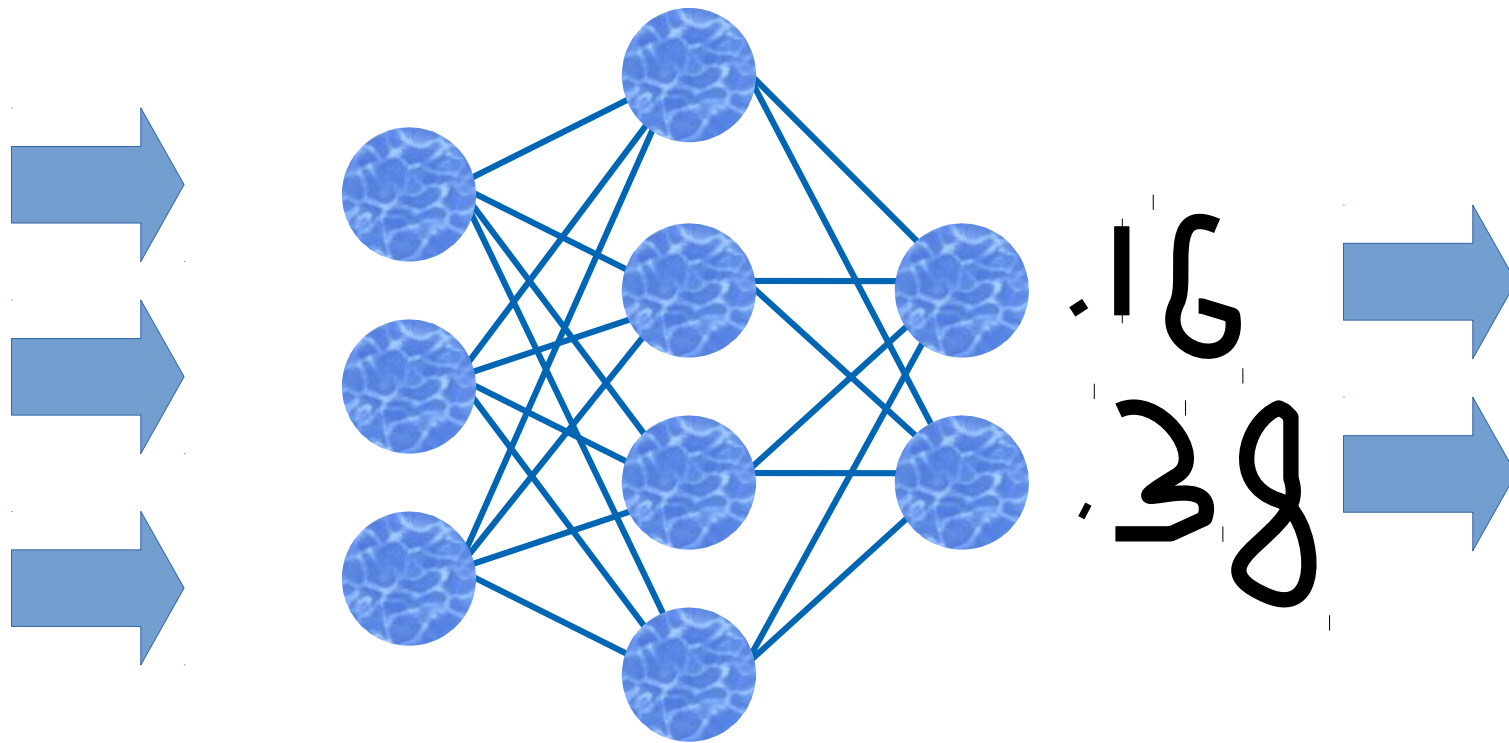
Remember, we found this one to be 0.16



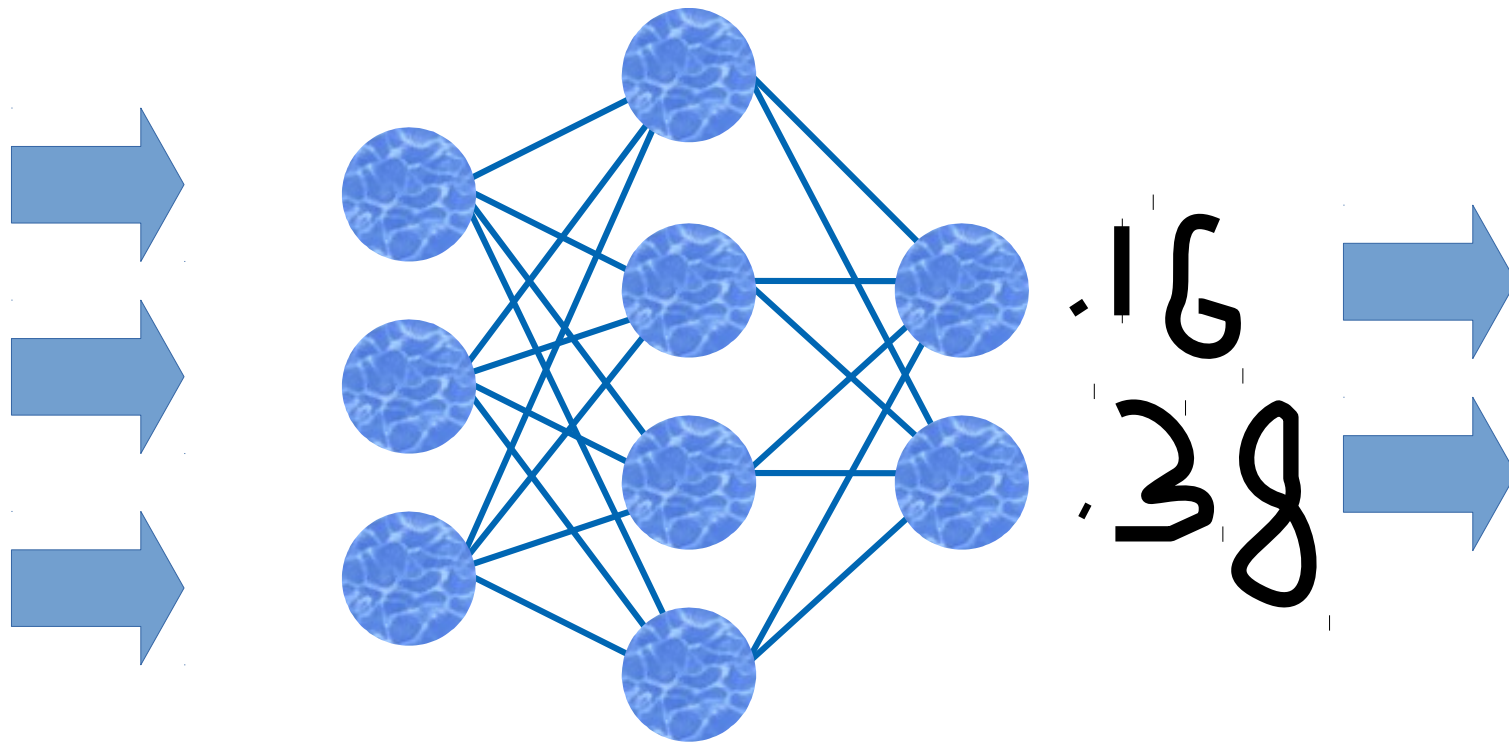


This one will also have a unique value

We are interpreting these numbers as probabilities that the inputs describe an observation of a corresponding class.



We are interpreting these numbers as probabilities that the inputs describe an observation of a corresponding class.



If something is either class one or class two, then the probabilities of each class should add up to 1

Looking at the numbers, $0.16 + 0.38$ is not 1.

So we do one last thing.

Looking at the numbers, $0.16 + 0.38$ is not 1.

So we do one last thing.

softmax

What is the softmax? Its just a function. A vector goes in, and a vector goes out. Only the vector that goes out always has a sum equal to 1.

$$Y_v = e^{X_v} / (\text{sum}_v(e^{X_v}))$$

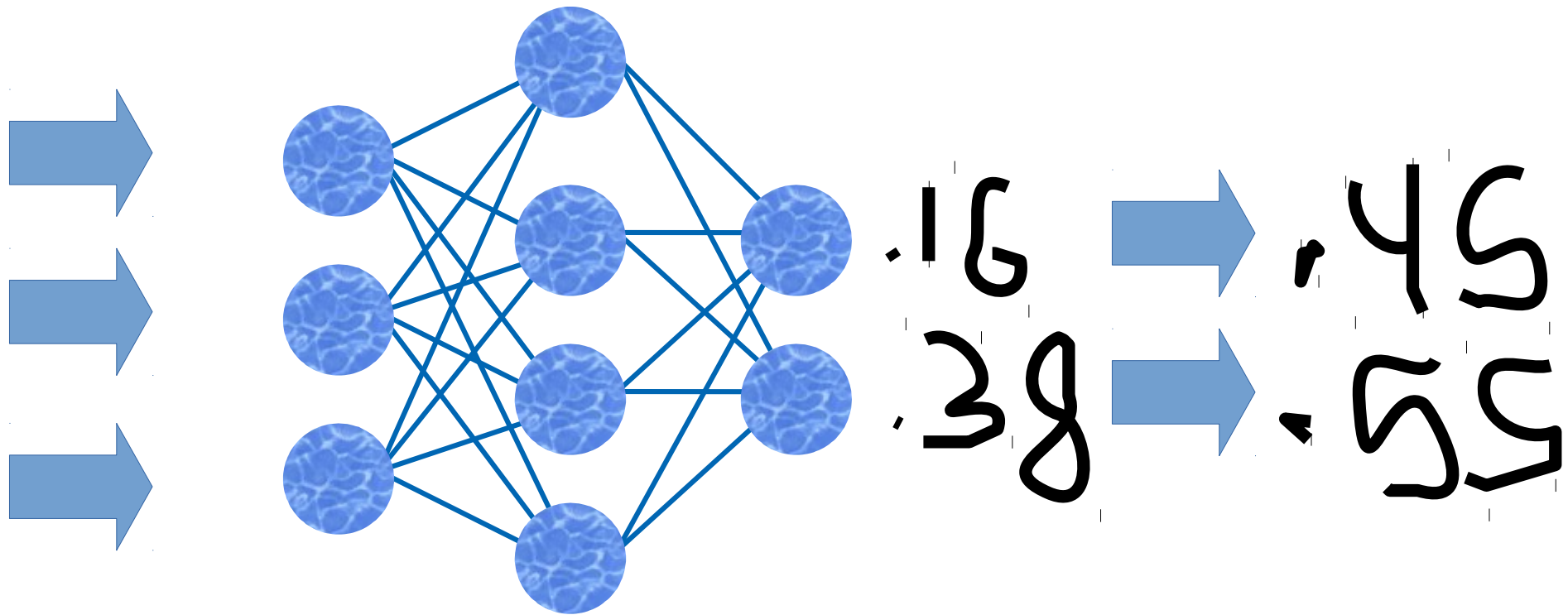
What is the softmax? Its just a function. A vector goes in, and a vector goes out. Only the vector that goes out always has a sum equal to 1.

$$Y_v = e^{X_v} / (\text{sum}_v(e^{X_v}))$$

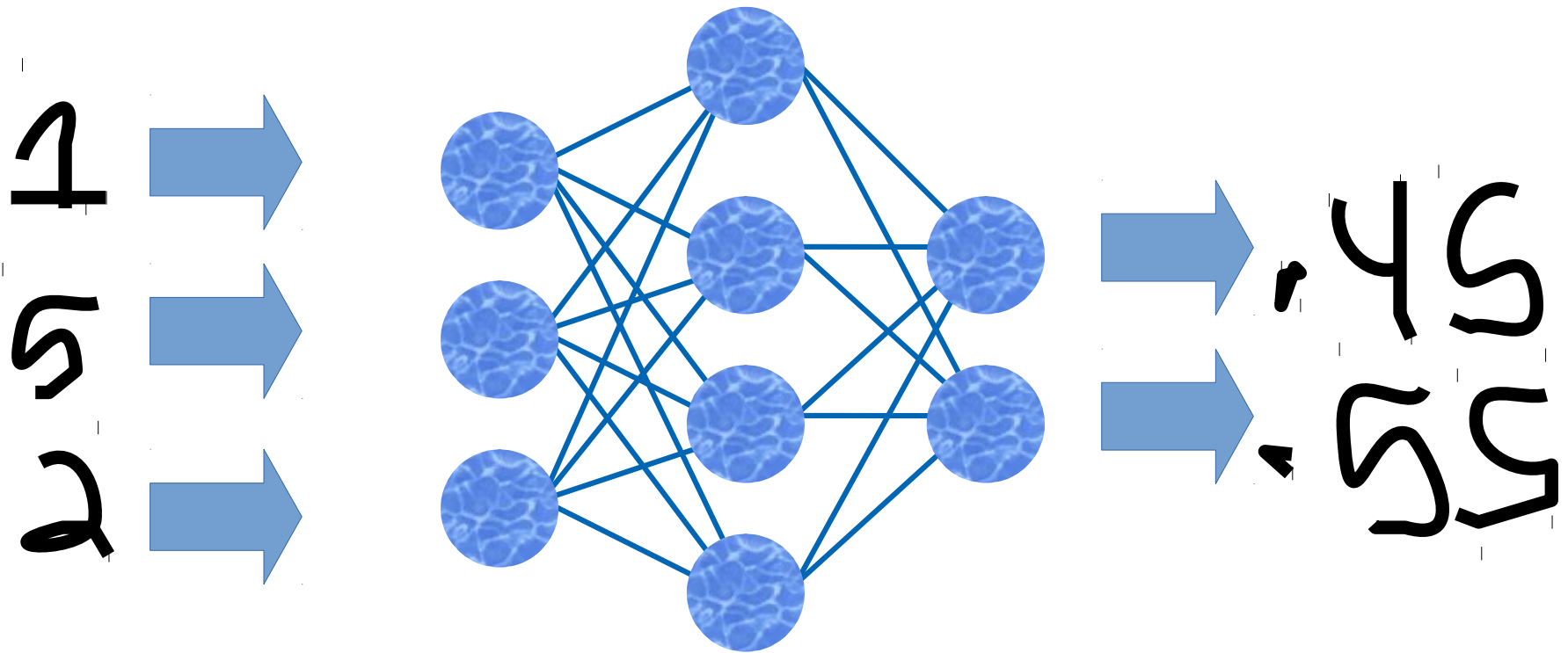
Vector in, vector out



Apply the softmax....

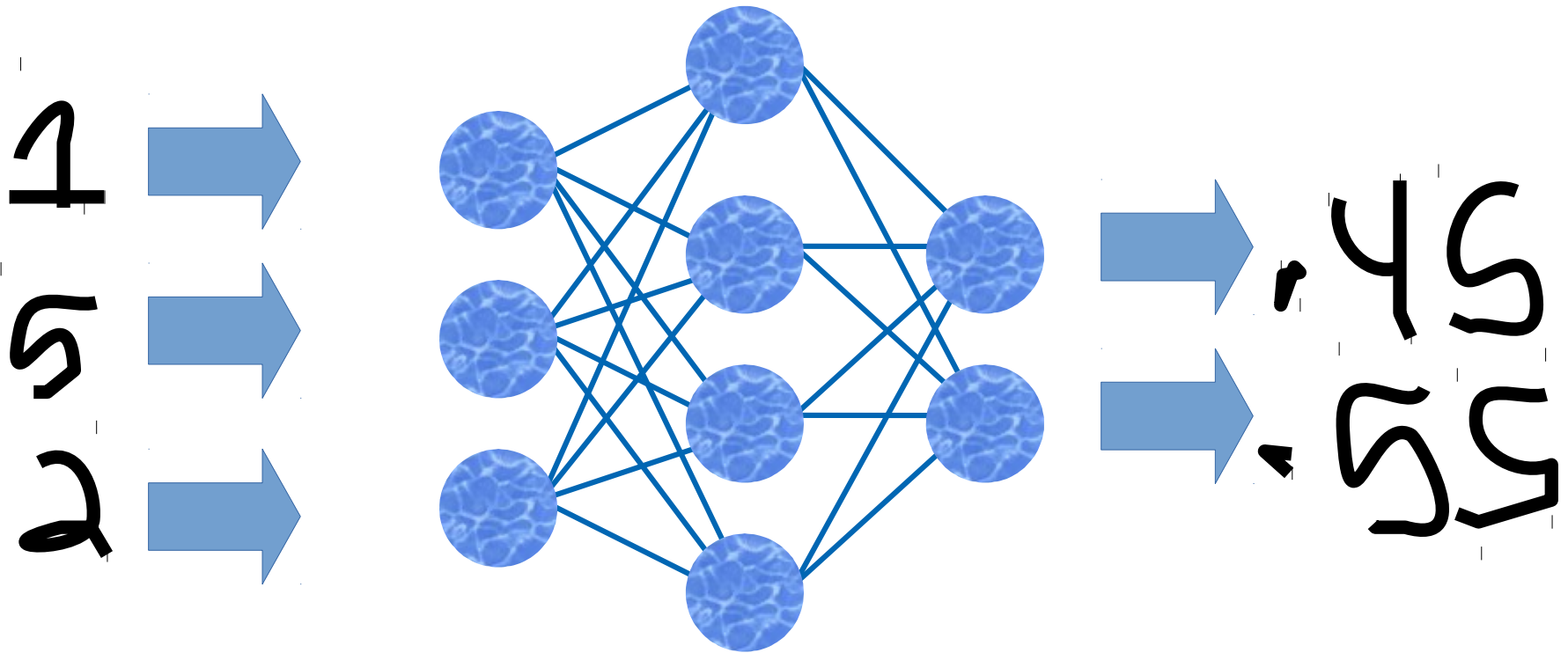


So, finally...

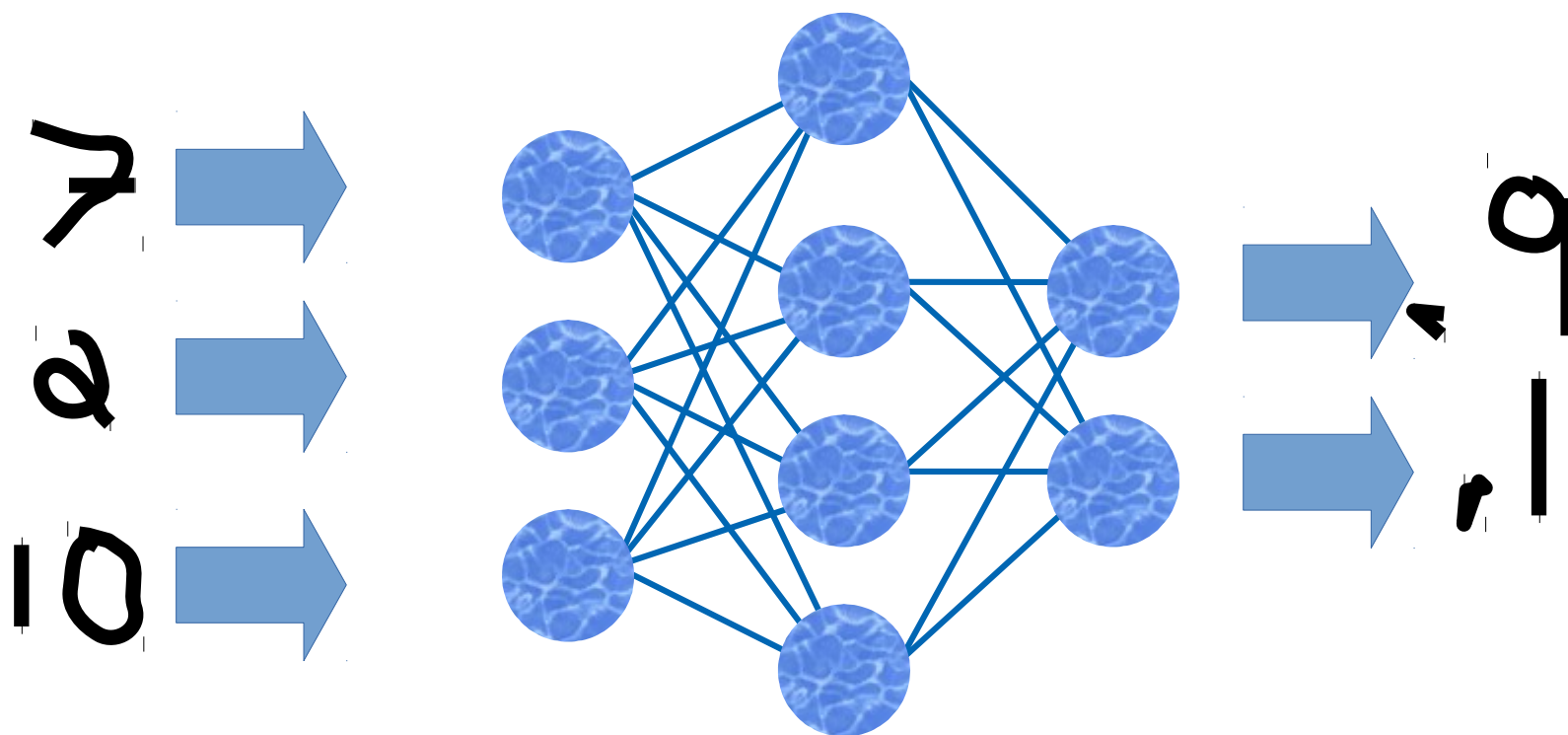


1, 5, and 2 are mapped to 0.45 and 0.55.

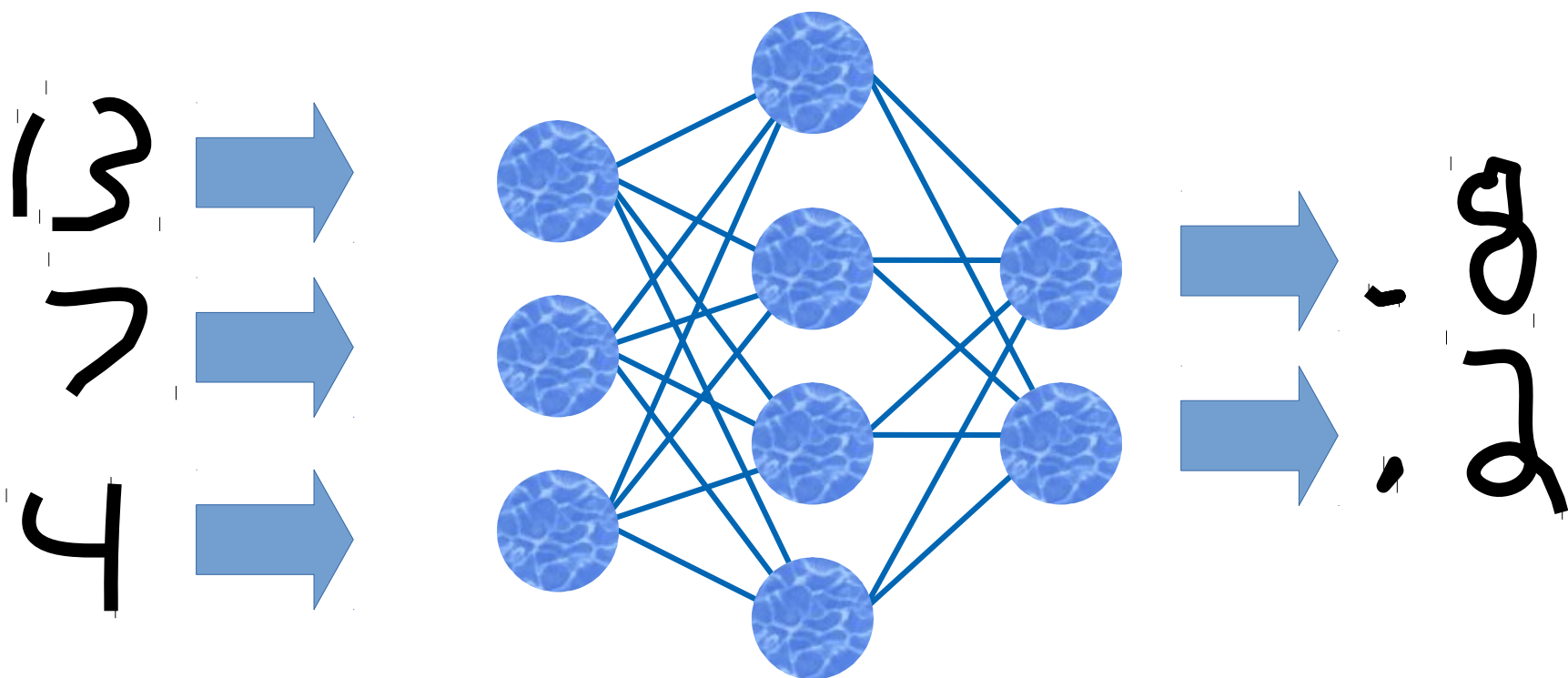
Meaning, the observation defined by the 3 input variables 1, 5, and 2 has a 0.45 chance of being class 1, and a 0.55 chance of being class 2.



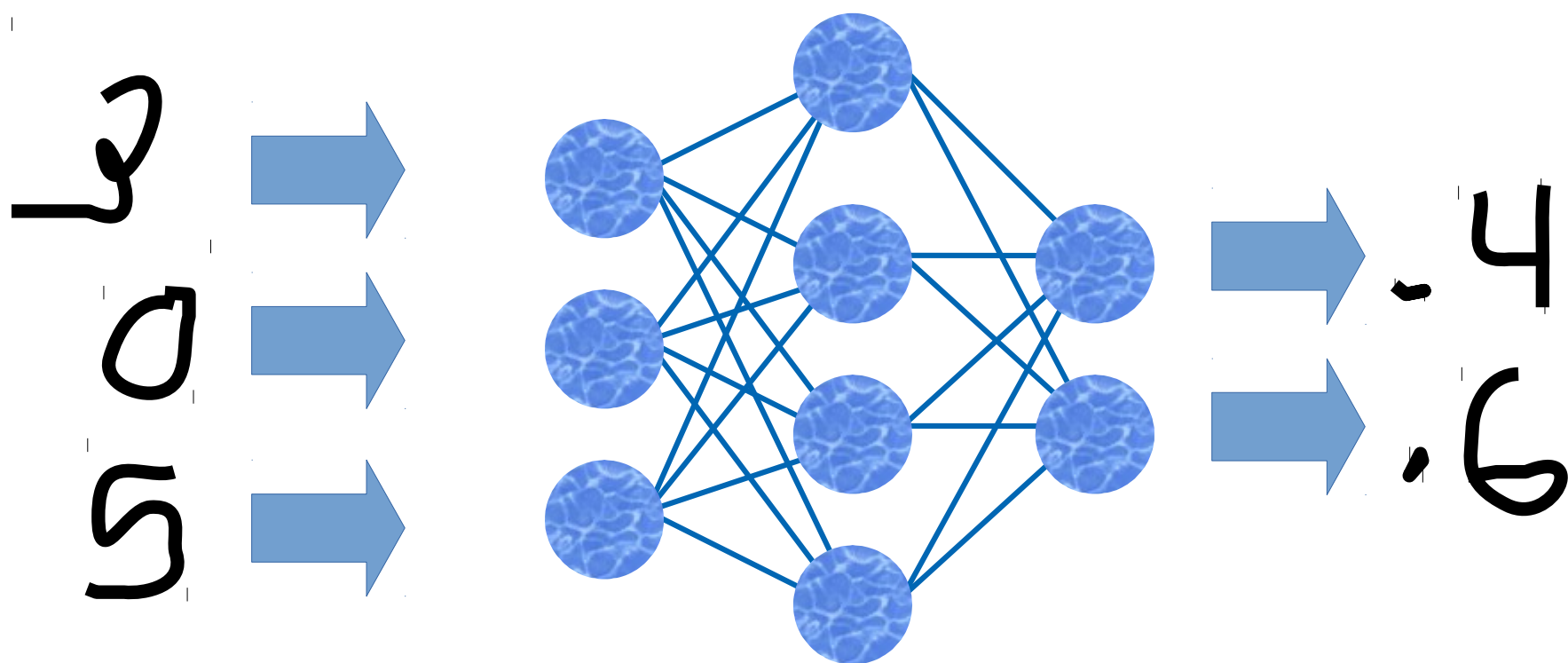
Different observations will yield different probabilities.



Different observations will yield different probabilities.

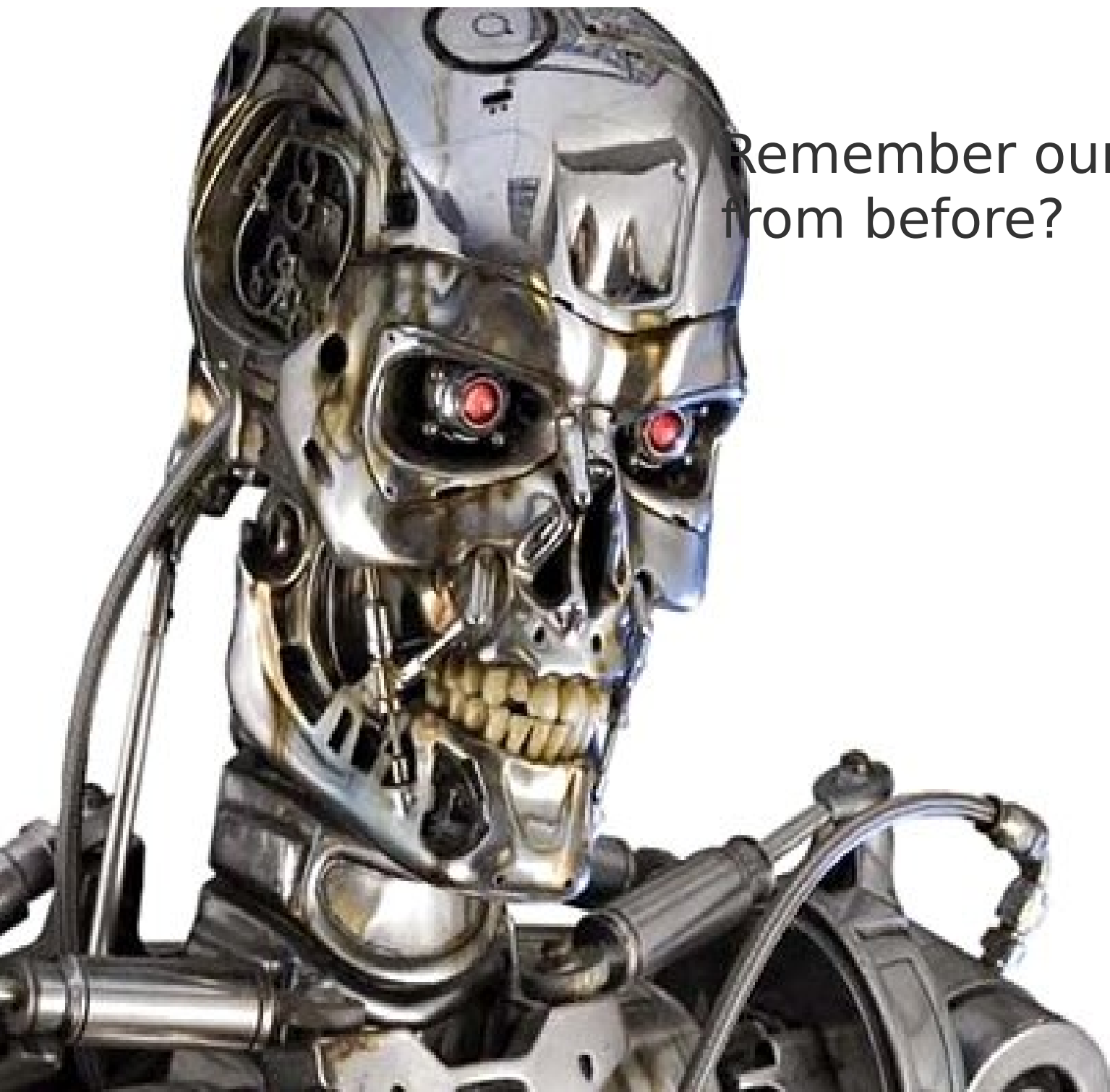


Different observations will yield different probabilities.



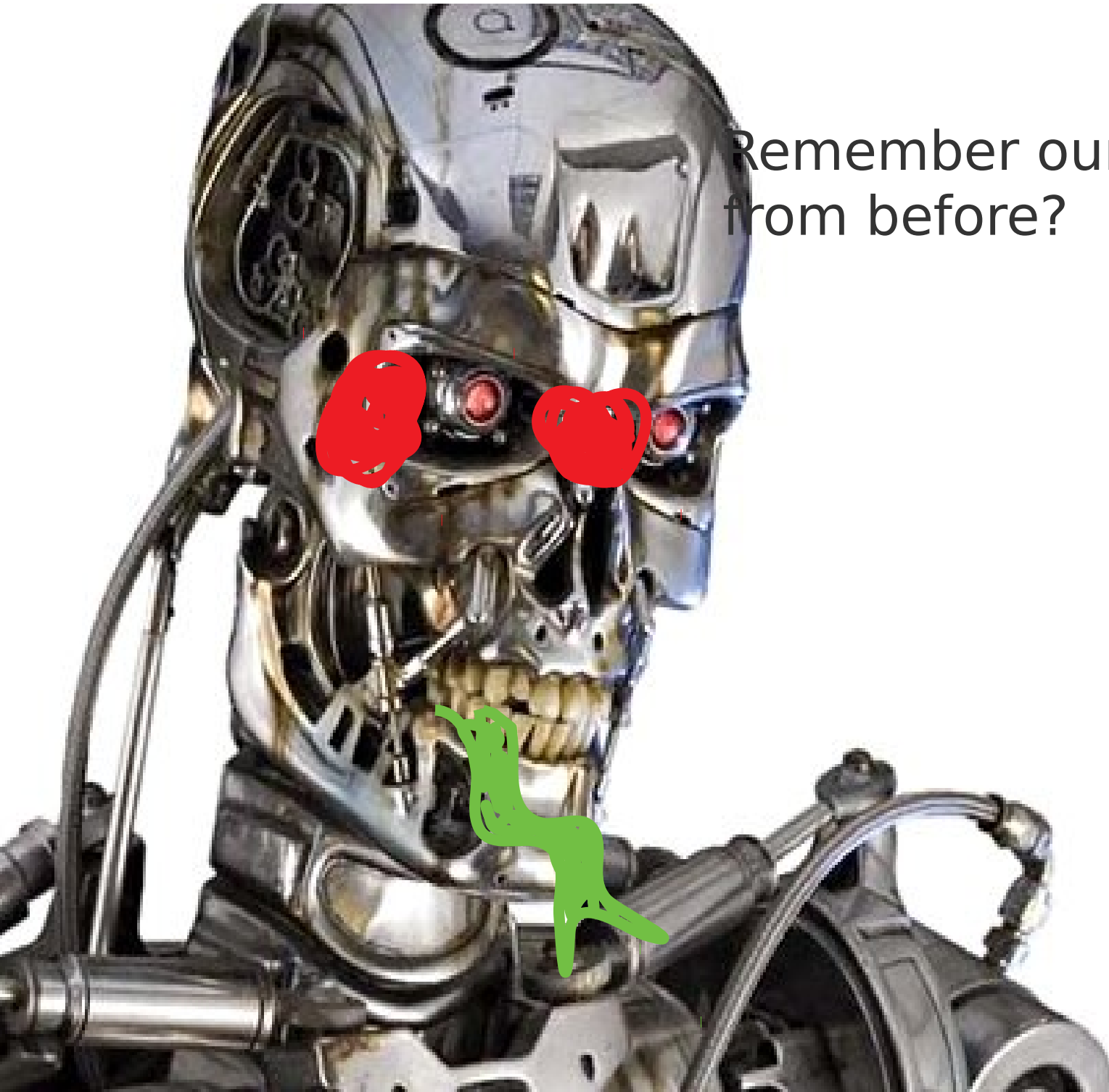
OK, so thats a forward pass.

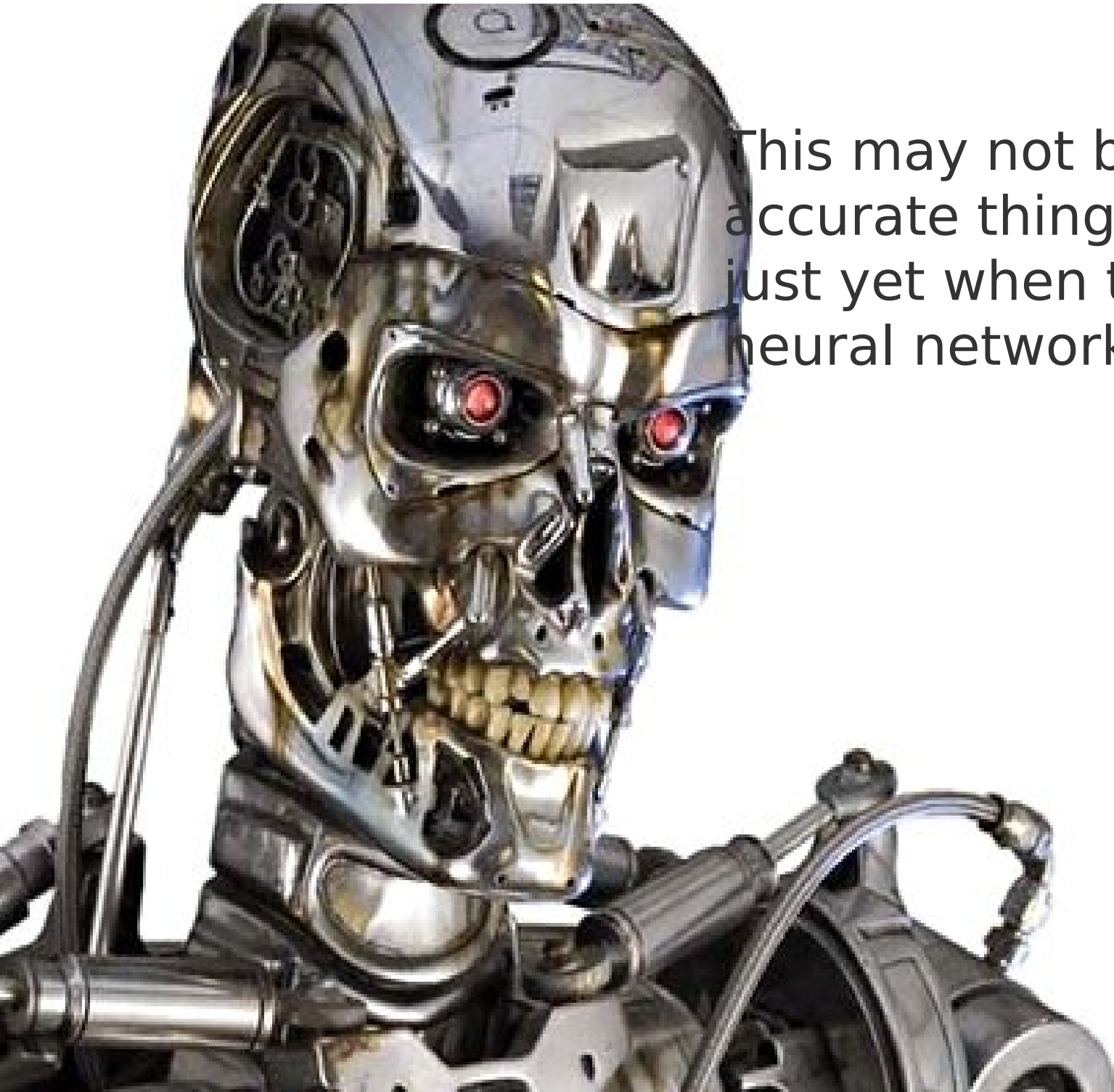
OK, so thats a forward pass. Need a break?
Questions?



Remember our evil robot
from before?

Remember our evil robot
from before?





This may not be the most accurate thing to be afraid of just yet when talking about neural network.

But that doesn't mean everything is perfect

Remember the three input variables from the example? They were

Number of Loans Paid Back

Number of Active Loans

Times moved in last 10 years

Remember the three input variables from the example? They were

Number of Loans Paid Back

Number of Active Loans

Times moved in last 10 years

So?

Lets say you apply for a loan from this bank and don't get it.

Lets say you apply for a loan from this bank and don't get it.

Money Please

Lets say you apply for a loan from this bank and don't get it.

The neural network predicted a very low probability that you would pay back the loan.

Dang

The bank used the neural network to decide you were not worth the risk based on these factors:

Number of Loans Paid Back

Number of Active Loans

Times moved in last 10 years

Remember doing the calculations for the forward pass, how the numbers get mixed up, combined, and fed through equations?

It might be hard to figure out exactly **why** you got such a low probability to pay back a loan.

You might want to ask, exactly how was I denied a loan? Which of these factors was the “deciding factor” if there was one? What do I need to do to improve my predicted probability so I can get one in the future?

?

?

?

With neural networks, it can be really hard to say.

This is one of the big challenges right now.

Dang

Interpreting neural networks more deeply will help us build better ones, use them most effectively, and ethically.

I see.

Even though neural nets have challenges, I think they are here to stay in many applications, and learning how they work can put you ahead of the game.

I hope this helped you decide if you want to learn more about these technologies in depth and start building valuable models yourself.

I hope this helped you decide if you want to learn more about these technologies in depth and start building valuable models yourself.

Since this is a class though, you have no choice and you have to build one on Friday. See you then.